

LES
CLÉFS
POUR

GFA BASIC 2 et 3 SUR ATARI

P.

FRANCK-OLIVIER
LELAIDIER

S.

LANGAGES

EDITIONS

PSI

I.

LANGAGE

***CLEFS POUR
GFA BASIC 2 ET 3
SUR ATARI***

Connaissez-vous la collection Matériels aux éditions PSI ?

Amiga

Super Jeux Amiga - Jean-François Sehan
102 programmes Amiga - Jacques Deconchat
Amiga en famille - Jean-François Sehan
Clefs pour Amiga - Benoît Michel, Vincent Labaye et Guy Herzet
Le livre de l'AmigaBasic - Benoît Michel et Jacques Boigontier

Amstrad

Super Jeux Amstrad - Jean-François Sehan
102 programmes Amstrad - Jacques Deconchat
Amstrad en famille - Jean-François Sehan
Basic Amstrad, méthodes pratiques - Jacques Boigontier et Bruno César
Périphériques et fichiers sur Amstrad CPC et PCW - Daniel-Jean David
CP/M Plus sur Amstrad - Yvon Dargery
Graphisme en assembleur sur Amstrad CPC - Francis Pierot
Clefs pour Basic 2 sur Amstrad PC - Augustin Garcia Ampudia

Atari ST

Super Jeux Atari ST, Basic GFA - Jean-François Sehan
102 programmes Atari ST, Basic GFA - Jacques Deconchat
Atari ST en famille, Basic GFA - Jean-François Sehan
Clefs pour Atari ST, nouvelle édition - Franck-Olivier Lelaidier
Atari ST efficace - Christophe Castro et Augustin Garcia Ampudia

PC

Super Jeux PC - Jean-François Sehan
102 programmes PC, GW-Basic et Basic2 - Jacques Deconchat
Clefs pour PC et compatibles, nouvelle édition avec DOS 3.3 - Daniel Martin, Guy Herzet et Philippe Jadoul

La loi du 11 mars 1957 n'autorisant, aux termes des alinéas 2 et 3 de l'article 41, d'une part, que les « copies ou reproductions strictement réservées à l'usage privé du copiste et non destinées à une utilisation collective », et, d'autre part, que les analyses et les courtes citations dans un but d'exemple et d'illustration, « toute représentation ou reproduction intégrale, ou partielle, faite sans le consentement de l'auteur ou de ses ayants droit ou ayants cause, est illicite » (alinéa 1^{er} de l'article 40).

Cette représentation ou reproduction, par quelque procédé que ce soit, constituerait donc une contrefaçon sanctionnée par les articles 425 et suivants du Code Pénal.

© Editions P.S.I. est une filiale de Nathan Editeur

GRUPPE
NATHAN
LA CITE

6-10, boulevard Jourdan, 75014 PARIS

1989

ISBN 2-86595-557-5

LANGAGE

CLEFS POUR GFA BASIC 2 ET 3 SUR ATARI

FRANCK-OLIVIER LELAIDIER



Présentation de l'auteur

Franck-Olivier Lelaidier : étudiant en informatique. Il écrit des articles pour divers magazines spécialisés en informatique.
Il est auteur aux Editions P.S.I. du livre "Clefs pour Atari ST".

Je remercie Augustin Garcia Ampudia pour sa collaboration.
Je dédie ce livre à David.

AVERTISSEMENT

Atari ST est une marque déposée d'Atari Corp.
GEM est une marque déposée de Digital Research.
GFA Basic est une marque déposée de Gfa Systemtechnik GmbH.

Ce livre n'est pas un manuel du GFA Basic et son contenu n'engage pas Gfa Systemtechnik GmbH. Il s'agit d'un ouvrage de référence complet, pratique et facile à utiliser.

Nous vous rappelons, par ailleurs, les termes de l'article 47 de la loi du 3 juillet 1985 :

"Toute reproduction autre que l'établissement d'une copie de sauvegarde par l'utilisateur, ainsi que toute utilisation d'un logiciel non expressément autorisée par l'auteur ou ses ayants droit, est passible des sanctions prévues par la dite loi".



A VOTRE SERVICE

DISQUETTE D'ACCOMPAGNEMENT

Certains ouvrages font l'objet d'une disquette d'accompagnement reprenant les programmes contenus dans le livre ou les applications associés. Pour les obtenir, reportez-vous à la page "disquette d'accompagnement" insérée au début de chaque ouvrage possédant une disquette.

EN COMPOSANT LE 3615 CODE OI * LIV NOTRE SERVICE MINTEL VOUS PROPOSE

- de vous renseigner sur notre catalogue et toutes nos nouveautés
- de vous indiquer le plus proche point de vente
- de répondre à vos questions techniques concernant nos ouvrages grâce à la messagerie P.S.I.

CATALOGUES ET "LIVRES MICRO"

Vous pouvez recevoir chez-vous les catalogues complets de nos ouvrages et être abonné gratuitement à la revue "Livres Micro". Pour ce faire, envoyez le coupon ci-dessous à :

Editions P.S.I. 6-10 boulevard Jourdan - 75014 PARIS



VOTRE AVIS NOUS INTERESSE

Je désire recevoir gratuitement : ☐ vos catalogues ☐ la revue "Livres Micro"

Pour nous permettre de faire de meilleurs livres, adressez-nous vos critiques et vos suggestions sur le présent ouvrage.

Titre de l'ouvrage : _____

- Ce livre vous donne-t-il toute satisfaction ?

- Y a-t-il un aspect du problème que vous auriez aimé voir aborder ?

Nom _____ Prénom _____ Age _____

Adresse _____

Profession _____

Centre d'intérêt : ☐ PC

☐ Macintosh

☐ Autre

PRESENTATION

A l'heure où cet ouvrage sortira des presses de l'imprimerie, plus de 20 000 logiciels GFA Basic 2.0 et 3.0 auront été vendus. L'engouement et le succès pour ce précieux outil de programmation réside dans sa simplicité de mise en oeuvre, sa vitesse fulgurante et ses innombrables possibilités aussi bien graphiques que structurelles. Le GFA Basic dans sa version 2.0 possède plus de deux cent cinquante instructions permettant des opérations aussi diverses que la manipulation des "lutins", de la souris, des fenêtres, ... ou que les appels systèmes au BIOS, XBIOS ou GEMDOS. La version 3.0 porte à plus de quatre cents le nombre des instructions et décuple à la fois possibilités et vitesse d'exécution. Tout (ou presque) devient possible à partir de ce Basic qui ressemble maintenant plus au langage Pascal ou au langage C qu'à un vulgaire interpréteur Basic. A l'instar du Pascal, le GFA Basic s'inspire du concept de programmation lisible et structurée : une seule instruction par ligne, des procédures et des fonctions comprenant variables locales et passage de paramètres par valeur ou par adresse. La lisibilité s'accroît par l'indentation automatique des lignes au fur et à mesure qu'elles sont comprises et validées par l'interpréteur. Une panoplie de définitions de types permet de se passer des déclarations de type à chaque fois que l'on tape une variable. En effet, l'interpréteur reconnaît les variables et leurs colle le suffixe adéquat automatiquement. De même, toute nouvelle variable ou procédure vous sera signalée : plus d'erreur de saisie possible.

Cet ouvrage a pour but de servir de référence pratique; vous trouverez toutes les instructions et les fonctions du GFA Basic décrites et illustrées par des programmes optimisés montrant les possibilités de l'interpréteur. Les instructions sont classées par thème, dans un ordre alphabétique qui vous facilitera les recherches. Deux index, un alphabétique de l'ensemble des instructions du GFA Basic 3.0 et un second sur seulement celles du GFA Basic 2.0 proposé sont à la fin du livre.

SOMMAIRE

SOMMAIRE

EDITEUR	13
LES MENUS DE L'EDITEUR	15
LES TOUCHES DE FONCTIONS	17
LES COMMANDES DE L'EDITEUR	17
COMMANDES ET INSTRUCTIONS DE BASE	19
LES DEFINITIONS DE TYPES	21
LES COMMANDES SUR LES FICHIERS	22
LES ENTREES ET LES EDITIONS	24
INSTRUCTIONS DIVERSES	29
OPERATEURS	33
LES OPERATEURS ARITHMETIQUES	35
LES OPERATEURS DE COMPARAISON	35
LES OPERATEURS LOGIQUES	35
VARIABLES, TABLEAUX ET POINTEURS	37
VARIABLES	39
DEFINITION DES TYPES DE VARIABLES	39
CONVERSIONS DE TYPE	40
GESTION DES VARIABLES	42
TABLEAUX	42
POINTEURS	44
FONCTIONS NUMERIQUES	51
FONCTIONS DE VARIABLES ENTIERES	53
INSTRUCTIONS BINAIRES	56
FONCTIONS LOGIQUES	57
FONCTIONS DE DECALAGES ET DE ROTATIONS	58
FONCTIONS DE MANIPULATIONS DE BITS	59
FONCTIONS DE VARIABLES REELLES	60
TRAITEMENT DES CHAINES DE CARACTERES	65
EXEMPLE DE TRAITEMENT DES CHAINES DE CARACTERES	73
PALINDROME ET ANAGRAMME	74
NOTATION POLONAISE INVERSE	78
INSTRUCTIONS DE STRUCTURES	83
GESTION DES ERREURS	93
GESTION DE LA MEMOIRE.....	97

HORLOGE ET INTERRUPTIONS	103
GESTION DU JOYSTICK	107
GESTION DES SONS	111
INTERFACAGE AVEC LES AUTRES LANGAGES	115
EXEMPLE DE COMMUNICATION : GFA BASIC ET ASSEMBLEUR	119
GESTION DU CLAVIER	129
GRAPHISME.....	133
PROGRAMMES GRAPHIQUES	147
TRACE D'HISTOGRAMMES ET DE CAMEMBERTS 2D ET 3D	149
UN MINI-PROGRAMME DE DESSIN	156
UNE TORTUE QUI DESSINE DES ARBRES	168
UN LUTIN AU BOUT DE LA SOURIS	171
DES ROUTINES A INTEGRER DANS VOS PROGRAMMES..	175
REMETTRE A JOUR LA PALETTE DES COULEURS....	175
QUELLE EST LA RESOLUTION DE L'ECRAN ?.....	176
QUELLE EST LA COULEUR D'UN REGISTRE ?.....	177
QUELLE EST L'ADRESSE DE L'ECRAN PHYSIQUE ?.	177
QUELLE EST L'ADRESSE DE L'ECRAN VIRTUEL ?....	177
LIGNE A	179
GESTION DES FENETRES, DES MENUS ET DE LA SOURIS	185
EXEMPLE DE PROGRAMMATION DES FENETRES ET DE LA SOURIS	197
FICHIERS ET REPERTOIRES	205
FICHIERS	207
REPERTOIRES	213
BIOS, XBIOS ET GEMDOS	217
LE BIOS	219
LES FONCTIONS DU BIOS ETENDU : XBIOS	224
LE GEMDOS	236
ROUTINES AES ET VDI	249

ANNEXES	271
TABLE DES CODES ASCII	273
TABLE DES CODES CLAVIER	274
TABLE DES CODES ESCAPES	275
RECAPITULATIF DES DEFINITIONS	277
MESSAGES D'ERREURS	282
COMPILATEUR DU GFA BASIC 2.02	284
 INSTRUCTIONS DU GFA 2.0	 287
 INDEX ALPHABETIQUE	 293

EDITEUR DU GFA BASIC

LES MENUS DE L'EDITEUR



Save : sauvegarde du programme en cours.

Load : chargement d'un programme à partir d'une unité de disque.

Save,A : sauvegarde sous forme de fichier ASCII (extension .LST) directement récupérable depuis un traitement de textes ou une base de données.

Merge : intégration au programme en cours d'un fichier ASCII (.LST), ce qui permet d'écrire un programme à partir de procédures sauvegardées précédemment.

Quit : met fin à la session de programmation en Basic GFA, et retour au bureau de GEM.

Llist : imprime le listing du programme en cours. Attention il ne fait pas de saut de page!

New : effacement du programme en cours.

Block : opérations sur les blocs. Appuyez sur la première lettre de l'option souhaitée ou cliquez dessus avec la souris.

Copy : copie le bloc sélectionné à partir de l'emplacement du curseur.

Move : déplace le bloc sélectionné à l'emplacement du curseur.

Write : écrit le bloc sélectionné sur disque.

Llist : imprime le bloc sélectionné.

Start : le curseur se positionne sur la marque de début de bloc.

End : le curseur se positionne sur la marque de fin de bloc.

^Del : détruit le bloc sélectionné.

Hide : désélectionne le bloc.

New : efface le programme courant de la mémoire.

Blk Sta : marque le début d'un bloc.

Blk End : marque la fin d'un bloc.

Replace : recherche et remplacement d'un mot ou d'une portion de mot.

Find : recherche d'un mot ou d'une portion de mot.

Pg up : monte d'une page-écran dans le programme.

Pg down : descend d'une page-écran dans le programme.

Text 16 ou Text 8 : affichage en 8 ou 16 pixels de hauteur de caractère.

Insert ou Overwrite : mode d'écriture en insertion ou en recouvrement.

Direct : passage en mode direct ; on revient en tapant sur la touche ESC en version 2.0 ou ed en 3.0.

Flip : passage sur la fenêtre de restitution.

Run : lancement d'un programme.

Test : regarde si tous les blocs, boucles ou procédures sont fermés correctement.



On passe dans ce nouveau menu en choisissant l'item sous le symbole Atari : "Editor" ou "Menu".

Save : revient à l'éditeur et demande le nom dans lequel il faut sauvegarder le programme courant.

Load : revient à l'éditeur et demande le nom du fichier qu'il faut charger.

Deflist : détermine le mode de visualisation (nous conseillons le mode 2).

Nouveaux noms : le GFA Basic vous demandera de confirmer toutes les nouvelles variables ou procédures que vous créez. Ainsi, il n'y aura plus de risque d'utiliser deux noms d'orthographes très voisines.

LES TOUCHES DE FONCTIONS

SHIFT-F1 : Save
F1 : Load
SHIFT-F2 : Save, A
F2 : Merge
SHIFT-F3 : Quit
F3 : Llist
SHIFT-F4 : New
F4 : Block
SHIFT-F5 : Blk Sta
F5 : Blk End
SHIFT-F6 : Replace
F6 : Find
SHIFT-F7 : Pg up
F7 : Pg down
SHIFT-F8 : Text 16/ Text 8
F8 : Insert / Overwrite
SHIFT-F9 : Direct
F9 : Flip
SHIFT-F10 : Run
F10 : Test

LES COMMANDES DE L'EDITEUR

^B : marque de début de bloc.
^C : passe à la page suivante.
^Delete : supprime la ligne du curseur.
^E : remplacement du texte.
^F : recherche d'un texte.
^Flèche bas : passe à la page-écran suivante.
^Flèche droite : place le curseur en fin de ligne.
^Flèche gauche : place le curseur en début de ligne.
^Flèche haut : revient à la page-écran précédente.
^Home : place le curseur au début du programme.

^K : marque de fin de bloc.

^N : insère une ligne.

^Q : appel du menu Bloc (identique à F4).

^R : revient à la page précédente.

^Tab : recule le curseur d'une tabulation.

^U : annule l'action précédente (version 3.0).

^Y : efface la ligne où se trouve le curseur.

^Z : saut en fin de programme.

Backspace : supprime le caractère précédent le curseur.

Delete : supprime le caractère sous le curseur.

Flèche bas : déplacement du curseur d'une ligne vers le bas.

Flèche droite : déplacement du curseur vers la droite.

Flèche gauche : déplacement du curseur vers la gauche.

Flèche haut : déplacement du curseur d'une ligne vers le haut.

Help : permet de replier ou de déplier une procédure.

Home : place le curseur au début de la page-écran.

Return ou Enter : passe à la ligne suivante.

Shift ^E : module de remplacement.

Shift ^F : module de recherche.

Tab : avance le curseur d'une tabulation.

Undo : annule l'action précédente (version 2.0).

Le signe ^ signifie qu'il faut appuyer sur la touche Control.

COMMANDES ET INSTRUCTIONS DE BASE

Dans tous les langages, il existe des instructions et des commandes que l'on ne peut classer dans aucune catégorie, cependant, il est important de bien les connaître.

LES DEFINITIONS DE TYPES

En début de programme, il est conseillé de définir toutes les variables avec leur type. Le GFA Basic autorise, dans une certaine mesure, ces définitions globales sans toutefois atteindre la précision des définitions des langages Pascal ou C. On remarquera que les instructions de définition de type ont parfois un comportement bizarre. Nous vous conseillons d'utiliser le mode DEFLIST 2 afin de voir clairement à l'affichage le suffixe de type de chaque variable.

Voici la liste des définitions possibles :

DEFBIT	DEFLIST
DEFBYT	DEFSTR
DEFFLT	DEFWRD
DEFINT	

DEFBIT ch\$

ch\$: expression alphanumérique.

Cette instruction définit une ou plusieurs variables booléennes.

DEFBYT ch\$

ch\$: expression alphanumérique.

Cette instruction définit une ou plusieurs variables entières codées sur un octet.

DEFFLT ch\$

ch\$: expression alphanumérique.

Cette instruction définit une ou plusieurs variables en virgule flottante codée sur huit octets.

DEFINT ch\$

ch\$: expression alphanumérique.

Cette instruction définit une ou plusieurs variables entières signées codées sur quatre octets, c'est-à-dire un mot long.

DEFLIST n

n : expression numérique entière.

Cette instruction définit le mode d'affichage des variables, des procédures et des fonctions.

0 --> les mots clefs du GFA Basic sont affichés en majuscules, les noms des variables, des procédures et des fonctions utilisateurs sont affichés en minuscules ;

1 --> les mots clefs du GFA Basic, les noms des variables, des procédures et des fonctions utilisateurs sont affichés avec leur première lettre en majuscules et le reste en minuscules ;

2 --> les mots clefs du GFA Basic sont affichés en majuscules, les noms des variables, des procédures et des fonctions utilisateurs sont affichées en minuscules avec le suffixe correspondant au type ;

3 --> les mots clefs du GFA Basic, les noms des variables, des procédures et des fonctions utilisateurs sont affichés avec leur première lettre en majuscules et le reste en minuscules. Le suffixe correspondant au type est placé par le GFA Basic.

DEFSTR ch\$

ch\$: expression alphanumérique.

Cette instruction définit une ou plusieurs variables alphanumériques.

DEFWRD ch\$

ch\$: expression alphanumérique.

Cette instruction définit une ou plusieurs variables entières signées codées sur deux octets, c'est-à-dire un mot.

LES COMMANDES SUR LES FICHIERS

Voici la liste des fonctions et des instructions répertoriées dans ce paragraphe :

CHAIN	PSAVE
EDIT	QUIT
LIST	RUN
LLIST	SAVE
LOAD	SYSTEM
NEW	

CHAIN fichier

fichier : expression alphanumérique.

Cette instruction procède au chargement d'un programme et à son exécution. Toutes les spécifications suivantes sont permises simultanément pour le nom de fichier :

- le nom de l'unité de disquettes (ou de disque dur).

Par exemple, CHAIN "B:PROGRAM.GFA" charge et exécute le programme PROGRAM.GFA de l'unité B ;

- CHAIN utilise également le système de dossiers, par exemple, CHAIN "dossier2\dossier1\PROGRAM.GFA" charge et exécute le programme PROGRAM.GFA du dossier1, contenu dans le dossier2 (indiqué par \, qui doit précéder tout nom de dossier).

EDIT

Cette instruction permet d'entrer dans le mode éditeur.

LIST fichier

fichier : expression alphanumérique.

Cette instruction provoque l'affichage du programme en cours à l'écran ou sur une disquette sous forme de fichier ASCII. Toutes les spécifications suivantes sont permises simultanément pour le nom de fichier :

- le nom de l'unité de disquettes (ou de disque dur).

Par exemple, LIST "B:PROGRAM.LIS" liste le programme en cours sur le fichier PROGRAM.LIS de l'unité B ;

- LIST utilise également le système de dossiers, par exemple, LIST "dossier2\dossier1\PROGRAM.LIS" liste le programme en cours sur le fichier PROGRAM.LIS du dossier1, contenu dans le dossier2 (indiqué par \, qui doit précéder tout nom de dossier).

LLIST

Imprime le listing du programme dans le format choisi grâce à DEFLIST. Attention, l'imprimante doit être impérativement reliée au système !

LOAD fichier

fichier : expression alphanumérique.

Cette instruction procède au chargement du fichier en mémoire centrale. Ce fichier doit être un fichier d'extension .GFA sauvegardé avec le Basic GFA. Toutes les spécifications suivantes sont permises simultanément pour le nom de fichier :

- le nom de l'unité de disquettes (ou de disque dur).

Par exemple, LOAD "B:PROGRAM.GFA" charge le programme PROGRAM.GFA de l'unité B ;

- LOAD utilise également le système de dossiers, par exemple, LOAD "dossier2\dossier1\PROGRAM.GFA" charge le programme PROGRAM.GFA du dossier1, contenu dans le dossier2 (indiqué par \, qui doit précéder tout nom de dossier).

NEW

Efface le programme qui se trouve en mémoire centrale et nettoie les variables.

PSAVE fichier

fichier : expression alphanumérique.

Cette instruction sauvegarde un programme avec un en-tête "auto-start", c'est-à-dire qu'il sera impossible de lister le programme puisque celui-ci se lancera automatiquement lors de son chargement avec l'instruction LOAD. Toutes les spécifications suivantes sont permises simultanément pour le nom de fichier :

- le nom de l'unité de disquettes (ou de disque dur).

Par exemple, PSAVE "B:PROGRAM.GFA" sauve le programme en cours sur le fichier PROGRAM.GFA de l'unité B ;

- PSAVE utilise également le système de dossiers, par exemple, PSAVE

"dossier2\dossier1\PROGRAM.GFA" sauve le programme en cours sur le fichier PROGRAM.GFA du dossier1, contenu dans le dossier2 (indiqué par \, qui doit précéder tout nom de dossier).

QUIT [n]

n : expression numérique entière.

Cette instruction provoque l'arrêt du programme et permet de quitter l'interpréteur directement sans passer par le menu.

Le paramètre est renvoyé au programme appelant (généralement le bureau de GEM).

RUN

Cette instruction exécute le programme en mémoire centrale.

SAVE fichier

fichier : expression alphanumérique.

Cette instruction sauvegarde le programme sur disque. Toutes les spécifications suivantes sont permises simultanément pour le nom de fichier :

- le nom de l'unité de disquettes (ou de disque dur);

Par exemple, SAVE "B:PROGRAM.GFA" sauve le programme en cours sur le fichier PROGRAM.GFA de l'unité B ;

- SAVE utilise également le système de dossiers, par exemple, SAVE "dossier2\dossier1\PROGRAM.GFA" sauve le programme en cours sur le fichier PROGRAM.GFA du dossier1, contenu dans le dossier2 (indiqué par \, qui doit précéder tout nom de dossier).

SYSTEM [n]

n : expression numérique entière.

Cette instruction provoque l'arrêt du programme et permet de quitter l'interpréteur directement sans passer par le menu.

Le paramètre est renvoyé au programme appelant (généralement le bureau de GEM).

LES ENTREES ET LES EDITIONS

Voici la liste des fonctions et des instructions répertoriées dans ce paragraphe :

CRSCOL	LOCATE
CRSLIN	LPOS
FORM INPUT	LPRINT
HTAB	MODE
INKEYS	OUT
INP	OUT?
INP?	POS
INPAUX	PRINT

INPMID	PRINT USING
INPUT	TAB
INPUT\$	VTAB
LINE INPUT	WRITE
CLS	

CRSCOL

Cette fonction renvoie le numéro de la colonne du curseur.

CRSLIN

Cette fonction renvoie le numéro de la ligne du curseur.

CLS

Cette instruction efface l'écran.

FORM INPUT n AS var

n : expression numérique entière ;

var : variable alphanumérique.

Cette instruction permet de modifier une chaîne de caractères limitée à un certain nombre de lettres spécifié par le paramètre "n" de manière semblable à l'instruction FORM INPUT. L'unique différence réside dans le fait que l'ancienne valeur de la chaîne est affichée avant la modification.

Par exemple, prenons A\$="ancienne".

L'instruction FORM INPUT 8 AS A\$ demande l'entrée d'une nouvelle chaîne et l'affecte à A\$.

FORM INPUT n,var

n : expression numérique entière ;

var : variable alphanumérique.

Cette instruction permet de saisir une chaîne de caractères limitée à un certain nombre de lettres spécifié par le paramètre "n". La cloche sonnera si l'on tente d'introduire plus de n caractères.

L'instruction FORM INPUT 8,A\$ saisit une chaîne contenant au plus 8 caractères.

HTAB colonne

colonne : expression numérique entière.

Cette instruction positionne le curseur sur la colonne précisée.

INKEY\$

Cette instruction permet de saisir un caractère au clavier.

A\$=INKEY\$ saisit un caractère au clavier et l'affecte à A\$.

INP(n)

n : expression numérique entière.

Cette fonction procède à la lecture sur un périphérique et attend la disponibilité de ce dernier, le cas échéant, les paramètres suivants sont acceptés :

- 0 = LST: l'imprimante ;
- 1 = AUX: la sortie série RS-232C ;
- 2 = CON: le clavier et l'écran ;
- 3 = MID: la prise MIDI.

Par exemple :

VOID INP(2) attend l'appui d'une touche indéfiniment.

INP?(n)

n : expression numérique entière.

Cette fonction renvoie l'état du port n choisi parmi les suivants :

- 0 = LST: l'imprimante ;
- 1 = AUX: la sortie série RS-232C ;
- 2 = CON: le clavier et l'écran ;
- 3 = MID: la prise MIDI.

La valeur renvoyée est soit -1, (TRUE) si un octet est disponible, soit 0, (FALSE) s'il ne l'est pas. Contrairement à la fonction INP, INP? n'attend pas !

Par exemple :

A=INP?(2) permet de savoir si une touche a été pressée.

INPAUX

Cette fonction renvoie la chaîne de caractères qui se trouve dans la mémoire tampon de l'interface série RS-232C.

INPMID

Cette fonction renvoie la chaîne de caractères qui se trouve dans la mémoire tampon MIDI.

INPUT [texte,(ou ;)]var1[,var2,...]

texte : expression alphanumérique ;

var1,var2,... : variables.

Cette instruction permet de saisir des données au clavier. Un texte de commentaires bien que facultatif éclaire l'utilisateur sur la nature de la requête. Si un point-virgule sépare le texte et les variables, l'affichage d'un point d'interrogation et d'un blanc suit celui du texte.

Par exemple :

INPUT "votre nom, votre âge";Nom\$,Age demande le nom et l'âge à l'utilisateur.

INPUT\$(n)

n : expression numérique entière.

Cette fonction renvoie les n caractères lus à partir du clavier. Le retour chariot est inutile.

Par exemple :

A\$=INPUT\$(8) affecte une chaîne de huit caractères à la variable A\$.

LINE INPUT [texte,(ou ;)] var1 [,var2...]

texte : expression alphanumérique ;

var1,var2,... : variables.

Cette instruction permet de saisir une ligne entière jusqu'au retour chariot (RETURN), autorisant ainsi les ponctuations telles que la virgule ou le point-virgule qui ne sont pas acceptées par l'instruction INPUT.

Par exemple :

LINE INPUT "Entrez ce qu'il vous plaît",A\$ laisse l'utilisateur totalement libre des caractères entrés.

LOCATE ligne,colonne

ligne,colonne : expressions numériques entières.

Cette instruction positionne le curseur à l'intersection de la ligne et de la colonne précisées.

LPOS(n)

n : expression numérique entière.

Cette fonction détermine la colonne à l'intérieur de la mémoire tampon de l'imprimante dans laquelle se trouve le curseur d'impression.

LPRINT [texte[,][;]]

texte : expression alphanumérique.

Cette instruction envoie un texte ou une expression à l'imprimante.

MODE n

n : expression numérique entière.

Cette instruction permet de choisir la manière d'afficher les nombres et les dates. Elle est prise en compte dans les instructions PRINT USING, DATE\$, SETTIME et FILES.

n :	format :	date :
0	#,###.##	jj.mm.aaaa
1	#,###.##	jj/mm/aaaa
2	#,###,##	jj.mm.aaaa
3	#,###,##	jj/mm/aaaa

OUT n,o
OUT #n,o

n : expression numérique entière ;

o : octet.

Cette instruction envoie un octet sur un périphérique donné, choisi parmi les suivants :

0 = LST: l'imprimante ;

1 = AUX: la sortie série RS232C ;

2 = CON: le clavier et l'écran ;

3 = MID: la prise MIDI.

La valeur du paramètre o est comprise entre 0 et 255. Si la valeur spécifiée est supérieure, le système prendra comme valeur (o MOD 256).

OUT?(n)

n : expression numérique entière.

Cette fonction renvoie l'état du port n choisi parmi les suivants :

0 = LST: l'imprimante ;

1 = AUX: la sortie série RS-232C ;

2 = CON: le clavier et l'écran ;

3 = MID: la prise MIDI.

La valeur renvoyée est soit -1, (TRUE) si un octet a été reçu, soit 0, (FALSE) s'il ne l'a pas été.

POS(n)

n : expression numérique entière.

Cette fonction renvoie la colonne de l'écran où se trouve le curseur.

PRINT [AT(colonne,ligne)][;][expression[,];]

colonne,ligne : expressions numériques entières ;

expression : expression.

Cette instruction affiche à la position courante de l'écran du texte ou une expression. Si le suffixe AT(colonne,ligne) suit l'instruction PRINT, le curseur est placé à la nouvelle position et l'affichage s'effectue à partir de celle-ci.

Par exemple :

PRINT AT 10,12;"BONJOUR" affiche le mot BONJOUR à partir de la 12ième ligne et de la 10^e colonne.**PRINT USING format,expression1[;expression2...]**

format : expression alphanumérique ;

expression1,expression2 : liste d'expression.

Cette instruction affiche des données dans un format précis :

remplace un chiffre ;

. détermine la position du point décimal ;

+ oblige l'affichage des signes "+" ;

— réserve de la place pour un signe "—" éventuel ;

* tous les 0 en tête de chaîne sont remplacés par des étoiles '*' ;

\$\$ affiche un \$ en tête de chaîne ;
 , affiche la virgule des milliers ;
 ^^^ format E+nn ;
 ^^^^ format E+nnn ;
 ! seul le premier caractère est affiché ;
 & l'intégralité de la chaîne de caractères est affichée ;
 \. \ affiche une chaîne dont la longueur est équivalente au nombre de points
 entre les deux séparateurs plus ceux-ci ;
 _ affiche le caractère suivant.

Par exemple :

PRINT USING "#####.##";102.541 affichera *102.54 .

TAB(n)

n : expression numérique entière.

Cette instruction provoque n tabulations. Si la position courante d'affichage se situe déjà sur la n° colonne, le curseur se place sur la n° colonne de la ligne suivante. La fonction TAB s'utilise toujours en conjonction avec l'instruction PRINT.

Par exemple :

PRINT TAB(10);"ici commence la dixième colonne" affiche le texte à partir de la dixième colonne.

VTAB ligne

ligne : expression numérique entière.

Cette instruction positionne le curseur sur la ligne donnée.

WRITE [expression[,][;]]

expression : expression.

Cette instruction affiche à la position courante de l'écran du texte ou une expression.

INSTRUCTIONS DIVERSES

Voici la liste des fonctions et des instructions répertoriées dans ce paragraphe :

CLEAR	OPTION BASE
CONT	PAUSE
DATA	READ
DEFNUM	REM
DELAY	RESTORE
END	STOP
HARDCOPY	VOID
LET	

CLEAR

Cette instruction efface toutes les variables et tous les tableaux de la mémoire. Il est interdit de la placer à l'intérieur d'une structure.

CONT

Cette commande directe relance l'exécution d'un programme après une instruction STOP ou un CONTROL-SHIFT-ALTERNATE (le STOP manuel).

DATA constante1[,constante2...]

constante1, constante2 : constantes.

Cette instruction stocke des données à l'intérieur d'un programme.

DEFNUM n

n : expression numérique entière.

Cette instruction formate les nombres affichés par PRINT. Seuls les n premiers chiffres d'un nombre sont affichés (le point décimal n'est pas pris en compte).

DELAY n

n : expression numérique entière.

Cette instruction arrête le programme pendant n secondes.

END

Cette instruction procède à la fermeture des fichiers ouverts et met fin à l'exécution du programme.

HARDCOPY

Cette instruction procède à une impression graphique de l'écran tout entier. Attention, il faut avoir préalablement défini le format à imprimer (voir bureau de GEM)!

LET var=expression

var : variable.

Cette instruction affecte une expression à une variable. Cette instruction est généralement omise.

OPTION BASE n

n : expression numérique entière.

Cette instruction détermine quel sera le premier indice de tous les tableaux (0 ou 1).

PAUSE n

n : expression numérique entière.

Permet d'inclure des pauses dans un programme. Le paramètre "n" indique le laps de temps pendant lequel le programme sera interrompu. Une seconde est équivalente à (n=50).

L'instruction PAUSE 3000 suspend l'exécution du programme pendant 1 minute.

READ var1[,var2...]

var1,var2... : variables.

Cette instruction lit les données stockées en DATA.

REM commentaires

' commentaires

Cette instruction permet d'inclure des commentaires dans un programme. On peut remplacer le REM par une apostrophe (').

RESTORE label

Cette instruction permet de rediriger le pointeur de lecture des données stockées en DATA vers une ligne précisée par le label.

STOP

Cette instruction interrompt l'exécution d'un programme. Seule l'instruction CONT permet de reprendre le cours de l'exécution.

VOID expression

~expression

expression : expression.

Permet de supprimer les variables inutiles que l'on affecte temporairement à une fonction du type INP ou FRE.

OPERATEURS

LES OPERATEURS ARITHMETIQUES

Le GFA Basic 3.0 possède les huit opérateurs arithmétiques suivants :

- + addition, par exemple : $3=2+1$;
- soustraction, par exemple : $5=8-3$;
- * multiplication, par exemple : $12=3*4$;
- / division, par exemple : $8=64/8$;
- signe négatif, par exemple : -7 ;
- ^ opérateur puissance, par exemple : $256=2^8$;
- DIV ou \ division entière, par exemple : $22 \text{ DIV } 3=22 \setminus 3=7$;
- MOD reste de la division entière (modulo), par exemple : $22 \text{ MOD } 7=1$.

LES OPERATEURS DE COMPARAISON

Le GFA Basic 3.0 dispose des six opérateurs de comparaison suivants :

- = égalité (à ne pas confondre avec celui d'affectation) ;
- < strictement inférieur ;
- > strictement supérieur ;
- =< ou <= inférieur ou égal ;
- => ou >= supérieur ou égal ;
- <> différent de.

Le résultat d'une comparaison est un booléen ; TRUE ou FALSE.

LES OPERATEURS LOGIQUES

Les variables booléennes possèdent leurs propres opérateurs; les opérateurs logiques :

AND (ET logique)

X	Y	X AND Y
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE

OR (OU logique)

X	Y	X OR Y
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

XOR (OU logique exclusif)

X	Y	X XOR Y
TRUE	TRUE	FALSE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

NOT (NON logique)

X	NOT X
TRUE	FALSE
FALSE	TRUE

IMP (implication logique)

X	Y	X IMP Y
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	TRUE
FALSE	FALSE	TRUE

EQV (équivalence logique)

X	Y	X EQV Y
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	TRUE

Table des priorités dans l'ordre décroissant :

()	les parenthèses ;
^	opérateur puissance ;
+ et -	signes ;
* et /	multiplication et division ;
DIV et MOD	division entière et modulo ;
+ et -	addition et soustraction ;
=, <>, <, >, <=, et >=	opérateurs de comparaison ;
AND, OR, XOR, NOT, IMP, et EQV	opérateurs logiques.

VARIABLES, TABLEAUX ET POINTEURS

VARIABLES

39

Le Basic GFA 3.0 reconnaît six types distincts de variables, les variables réelles, les variables entières sur un, deux ou quatre octets, et enfin les variables booléennes et alphanumériques. Les noms de variables doivent toujours commencer par une lettre et comportent un nombre quelconque de caractères (il est toutefois conseillé d'éviter les noms trop longs) d'un seul tenant et tous significatifs. Le type de chaque variable est indiqué par une déclaration préalable (voir le chapitre consacré à l'éditeur) ou bien par un suffixe explicite qui prévaut sur toutes les définitions globales antérieurement déclarées. Utilisez systématiquement les types de variables les plus adaptés à votre problème et ne vous laissez pas tenter par les conversions automatiques effectuées par l'interpréteur, ces facilités vous pénaliseront lourdement en vous empêchant d'évoluer vers une programmation structurée ou d'adapter vos programmes dans d'autres langages (C ou Pascal).

Définition des types de variables

Les variables réelles

Les variables réelles (ou Float) comportent le suffixe distinctif # (ou aucun suffixe) ou sont déclarées par l'instruction DEFLOAT. Elles sont mémorisées sur huit octets en virgule flottante. La représentation des variables réelles s'effectue dans un intervalle allant de -2.225073858507E-308 à +3.5953862697251E+308.

Les variables entières

Les variables entières se subdivisent en trois catégories :

- les variables entières signées du type Integer, qui comportent le suffixe distinctif % ou sont déclarées par l'instruction DEFINT. Elles sont mémorisées sur quatre octets. La représentation de ces entiers s'effectue dans un intervalle allant de -2147483648 à +2147483647 ;
- les variables entières signées du type Word, qui comportent le suffixe distinctif & ou sont déclarées par l'instruction DEFWORD. Elles sont mémorisées sur deux octets. La représentation de ces entiers s'effectue dans un intervalle allant de -32768 à +32767 ;
- les variables entières non signées du type Byte, qui comportent le suffixe distinctif ! ou sont déclarées par l'instruction DEFBYTE. Elles sont mémorisées sur un octet. La représentation de ces entiers s'effectue dans un intervalle allant de 0 à 255.

Les variables booléennes

Les variables booléennes (ou Boolean) comportent le suffixe distinctif ! ou sont déclarées par l'instruction DEFBIT. Elles sont mémorisées sur un octet. Ces variables ne peuvent prendre que deux valeurs : TRUE (vrai, codé par la valeur -1) ou FALSE (faux, codé par la valeur 0).

Les variables alphanumériques

Les variables alphanumériques (ou String) comportent le suffixe distinctif \$ ou sont déclarées par l'instruction DEFSTR. Ces variables contiennent au plus 32767 caractères, chaque caractère étant codé sur un octet. Leur manipulation interne nécessite l'emploi d'un descripteur, en particulier pour connaître la longueur de la chaîne.

Chacun de ces différents types accepte la structure tableau (ou champs) avec un nombre quelconque de dimensions, la seule restriction provenant de la mémoire disponible.

Conversions de type

Voici les fonctions traitées dans cette section :

CINT	CVS	MKS\$
CFLOAT	HEX\$	OCT\$
CVD	MKD\$	VAL
CVF	MKF\$	VAL?
CVI	MKI\$	
CVL	MKL\$	

CINT(x)

x : expression numérique.

Convertit le nombre en virgule flottante x en une valeur entière arrondie.

CFLOAT(x)

x : expression numérique.

Convertit une valeur entière x en un nombre en virgule flottante.

CVD(ch\$)

ch\$: expression alphanumérique.

Convertit une chaîne de huit caractères en une valeur numérique GFA Basic 3.0.

CVF(ch\$)

ch\$: expression alphanumérique.

Convertit une chaîne de six caractères contenant un nombre au format GFA Basic 2.0 (et antérieurs) en une valeur numérique réelle GFA Basic 3.0.

CVI(ch\$)

ch\$: expression alphanumérique.

Convertit une chaîne de deux caractères en mot (16 bits).

CVL(ch\$)

ch\$: expression alphanumérique.

Convertit une chaîne de quatre caractères en mot long (32 bits).

CVS(ch\$)

ch\$: expression alphanumérique.

Convertit une chaîne de quatre caractères contenant un nombre au format Basic Atari en une valeur numérique réelle GFA Basic.

HEX\$(n)

n : expression numérique entière.

Transforme la valeur n en une chaîne de caractères correspondant à la valeur hexadécimale (base 16) de n.

Par exemple :

PRINT HEX\$(-23) affiche FFFFFFFE9.

PRINT HEX\$(26359) affiche 66F7.

MKD\$(x)

x : expression numérique.

Convertit x au format MBasic sur huit octets.

MKF\$(x)

x : expression numérique.

Convertit x au format GFA Basicsur six octets.

MKI\$(n)

n : expression numérique entière.

Convertit un entier 16 bits en une chaîne de deux caractères.

MKL\$(n)

n : expression numérique entière.

Convertit un entier long 32 bits en une chaîne de quatre caractères.

MKS\$(x)

x : expression numérique.

Convertit x au format BASIC ST sur quatre octets.

OCT\$(n)

n : expression numérique entière.

Transforme la valeur n en une chaîne de caractères correspondant à la valeur octale de n (base 8).

Par exemple :

PRINT OCT\$(456) affiche 710 .

VAL(ch\$)

ch\$: expression alphanumérique.

Transforme la chaîne de caractères en une valeur numérique (si cela est possible !).

VAL?(ch\$)

ch\$: expression alphanumérique.

Renvoie le nombre de caractères de la chaîne pouvant être représentés par une valeur numérique.

Par exemple :

PRINT VAL?("&367+25") affiche 3.

Gestion des variables**CLR var1[,var2,...]**

var1,var2 : variables numériques.

Cette instruction efface la suite spécifiée de variables, mais pas les tableaux !

SWAP var1,var2

var1,var2 : variables numériques.

Cette instruction procède à l'échange des contenus des variables var1 et var2, à condition que ces variables soient de même type.

TABLEAUX

Voici les fonctions traitées dans cette section :

ARRAYFILL	INSERT
DELETE	QSORT
DIM	SSORT
DIM?	SWAP
ERASE	

ARRAYFILL tab(),n

tab : tableau numérique.

Cette instruction permet de remplir l'intégralité d'un tableau numérique avec la valeur n.

DELETE tab(i)

i : expression numérique entière ;

tab : tableau.

Cette instruction détruit l'élément d'indice i du tableau numérique ou alphanumérique considéré. Tous les éléments d'indice supérieur sont décalés vers le bas, c'est-à-dire que l'élément d'indice i+1 deviendra l'élément d'indice i, l'élément d'indice i+2 deviendra l'élément d'indice i+1...

Voir également l'instruction **INSERT**.

DIM tab1(i11[,i12,...]),tab2(i21[,i22,...])...

tab1,tab2... : tableaux ;

i11,i12...i21,i22... : expressions numériques entières.

Cette instruction définit les dimensions d'un ou plusieurs tableaux. Les paramètres représentent les dimensions d'un tableau : i11 pour la première dimension du premier tableau...

Voir également la fonction **DIM?**.

DIM?(tab())

tab : tableau.

Cette fonction renvoie le nombre d'éléments d'un tableau donné.

Voir également l'instruction **DIM**.

ERASE tab1(),tab2(),...

tab1,tab2... : tableaux.

Cette instruction efface de la mémoire les tableaux indiqués (équivalent à l'instruction **CLR** pour les variables).

INSERT tab(i)=exp

i : expression numérique entière ;

exp : expression ;

tab : tableau.

Cette instruction insère l'expression exp dans le tableau numérique ou alphanumérique considéré au rang d'indice i. Tous les éléments sont décalés vers le haut, c'est-à-dire que l'élément d'indice i deviendra l'élément d'indice i+1, l'élément d'indice i+1 deviendra l'élément d'indice i+2...

Voir également l'instruction **DELETE**.

QSORT tab1(ordre)[,n][tab2()]

QSORT tab1(ordre) **WITH** tab()[,n][tab2()]

tab1 : tableau ;

tab2 : tableau de mots longs ;

tab : tableau numérique entier ;

n : expression numérique entière ;

ordre : signe +, - ou rien.

Cette instruction trie un tableau de type quelconque dans l'ordre indiqué (croissant = +, décroissant = -) selon la méthode "Quick Sort" ; si celui-ci est omis, l'ordre choisi par défaut sera l'ordre croissant. Le paramètre optionnel n indique que seuls les n premiers éléments doivent être triés en commençant par le premier (indice 0 ou indice 1 selon **OPTION BASE**). La clause **WITH** ne fonctionne que sur les tableaux de chaînes; le tableau tab comporte au moins deux cent cinquante six éléments et indique la table d'ordre (la table ASCII est prise par défaut lorsque l'instruction **WITH** n'est pas spécifiée). La compréhension de l'usage de cette table est délicate. En effet, chaque caractère se voit affecter un ordre de priorité, c'est-à-dire sa position dans la nouvelle table. En clair, le code ASCII sert d'indice pour retrouver la priorité d'un caractère (dans la table ASCII, la valeur et l'indice sont identiques).

Voir également l'instruction **SSORT**.

SSORT tab1(ordre)[,n][tab2()
SSORT tab1(ordre) **WITH** tab()[,n][tab2()]

tab1 : tableau ;
 tab2 : tableau de mots longs ;
 tab : tableau numérique entier ;
 n : expression numérique entière ;
 ordre : signe +, - ou rien.

Cette instruction trie un tableau de type quelconque dans l'ordre indiqué (croissant = +, décroissant = -) selon la méthode "Shell Sort" ; si celui-ci est omis l'ordre choisi par défaut sera l'ordre croissant. Le paramètre optionnel n indique que seuls les n premiers éléments doivent être triés en commençant par le premier (indice 0 ou indice 1 selon **OPTION BASE**). La clause **WITH** ne fonctionne que sur les tableaux de chaînes ; le tableau tab comporte au moins deux cent cinquante six éléments et indique la table d'ordre (la table ASCII est prise par défaut lorsque l'instruction **WITH** n'est pas spécifiée). La compréhension de l'usage de cette table est délicate. En effet, chaque caractère se voit affecter un ordre de priorité, c'est-à-dire sa position dans la nouvelle table. En clair, le code ASCII sert d'indice pour retrouver la priorité d'un caractère (dans la table ASCII, la valeur et l'indice sont identiques).

Voir également l'instruction **QSORT**.

SWAP tab1(),tab2()

tab1,tab2 : tableaux.

Procède à l'échange des contenus des tableaux tableau1() et tableau2(), à condition que ces tableaux soient de même type. L'échange est très rapide, puisque seuls les descripteurs sont échangés.

POINTEURS

Grâce à l'instruction **VARPTR** (ou **V:**), l'accès direct en mémoire d'une variable est rendu possible : **VARPTR**(var) renvoie l'adresse du premier octet de la zone-mémoire où celle-ci est mémorisée. Les chaînes de caractères et les tableaux possèdent un descripteur dont l'adresse est donnée par **ARRPTR**.

Le descripteur d'une chaîne se compose de six octets ; les quatre premiers fournissent l'adresse du premier caractère de la chaîne et les deux derniers indiquent la taille de la chaîne. Derrière la chaîne proprement dite se trouve un éventuel octet de remplissage (pour que la longueur soit paire) suivi de l'adresse de début du descripteur correspondant.

Le descripteur des tableaux de variables réelles, entières ou booléennes est similaire. Les quatre premiers octets fournissent l'adresse du début du tableau, tandis que les deux derniers indiquent le nombre de dimensions du tableau. Au début du tableau, le nombre d'éléments de chaque dimension est spécifié sur quatre octets en commençant par la dernière dimension. Ensuite viennent les éléments eux-mêmes qui occupent respectivement huit octets pour les variables

réelles, quatre pour les variables entières et un seul bit pour les variables booléennes.

Le descripteur des tableaux de chaînes de caractères se décompose également en six octets. Les quatre premiers octets fournissent l'adresse du début du tableau, tandis que les deux derniers indiquent le nombre de dimensions du tableau. Au début du tableau, le nombre d'éléments de chaque dimension est spécifié sur quatre octets en commençant par la dernière dimension. Ensuite viennent non pas les éléments, mais les descripteurs de chaque élément dans l'ordre des indices, identiques à ceux indiqués pour les variables simples, et dans chacun de ces descripteurs se trouve l'adresse respective de l'élément en question.

Voici les fonctions et les instructions traitées dans cette section :

*	DPEEK	POKE
{}	DPOKE	SDPOKE
ABSOLUTE	FLOAT{}	SINGLE{}
ARRPTR	INT{}	SLPOKE
BYTE{}	LONG{}	SPOKE
CARD{}	LPEEK	TYPE
CHAR{}	LPOKE	V:
DOUBLE{}	PEEK	VARPTR

ABSOLUTE var,adr

var : variable ;

adr : mot long.

Cette instruction modifie l'emplacement de la variable indiquée. Elle la place à partir de l'adresse adr. La variable peut être d'un type quelconque, sauf de type tableau.

ARRPTR(var)

*var

var : variable.

Cette fonction renvoie l'adresse du descripteur d'un tableau ou d'une chaîne de caractères. Pour les autres variables, **ARRPTR** et **VARPTR** renvoient le même résultat.

Voir également la fonction **VARPTR**.

BYTE{adr}

adr : mot long ;

Cette instruction écrit un octet à l'adresse adr.

Voir également l'instruction **POKE**.

BYTE{adr}

adr : mot long.

Cette fonction lit un octet à l'adresse adr.

Voir également l'instruction **PEEK**.

CARD{adr}

adr : mot long.

Cette instruction écrit un mot aux adresses adr et adr+1.

Voir également l'instruction **DPOKE**.

CARD{adr}

adr : mot long.

Cette fonction lit un mot aux adresses adr et adr+1.

Voir également l'instruction **DPEEK**.

CHAR{adr}

adr : mot long.

Cette instruction écrit une chaîne de caractères terminée par un octet nul à partir de l'adresse adr.

CHAR{adr}

adr : mot long.

Cette fonction lit une chaîne de caractères terminée par un octet nul à partir de l'adresse adr.

DOUBLE{adr}

adr : mot long.

Cette instruction écrit un nombre en virgule flottante au format IEEE (sur huit octets) à partir de l'adresse adr.

DOUBLE{adr}

adr : mot long.

Cette fonction lit un nombre en virgule flottante au format IEEE (sur huit octets) à partir de l'adresse adr.

FLOAT{adr}

adr : mot long.

Cette instruction écrit un nombre en virgule flottante sur huit octets à partir de l'adresse adr.

FLOAT{adr}

adr : mot long.

Cette fonction lit un nombre en virgule flottante sur huit octets à partir de l'adresse adr.

INT{adr}

adr : mot long.

Cette instruction écrit un mot signé (en complément à 2, entre -32768 et +32767) aux adresses adr et adr+1.

Voir également l'instruction **DPOKE**.

INT{adr}

adr : mot long.

Cette fonction lit un mot signé (en complément à 2, entre -32768 et +32767) aux adresses adr et adr+1.

Voir également l'instruction **DPEEK**.

LONG{adr}

{adr}

adr : mot long.

Cette instruction écrit un mot long signé (en complément à 2, entre -2147483648 et +2147483647) aux adresses adr, adr+1, adr+2 et adr+3.

Voir également l'instruction **LPOKE**.

LONG{adr}

{adr}

adr : mot long.

Cette fonction lit un mot long signé (en complément à 2, entre -2147483648 et +2147483647) aux adresses adr, adr+1, adr+2 et adr+3.

Voir également l'instruction **LPEEK**.

PEEK(adr)**DPEEK(adr)****LPEEK(adr)**

adr : mot long.

Ces fonctions renvoient respectivement :

- l'octet contenu à l'adresse adr ;
- le mot contenu aux adresses adr et adr+1 ;
- le mot long signé (en complément à 2, entre -2147483648 et +2147483647) contenu aux adresses adr, adr+1, adr+2, et adr+3.

Tous les "Peek" sont exécutés en mode superviseur.

Voir également les fonctions **BYTE{}**, **CARD{}**, **INT{}**, **LONG{}** et **{}**.

POKE adr,n**DPOKE adr,n****LPOKE adr,n**

adr : mot long ;

n : expression numérique entière.

Ces instructions inscrivent respectivement :

- l'octet n (0-255) à l'adresse adr ;
- le mot n (0-65535) aux adresses adr et adr+1. L'adresse adr doit obligatoirement être paire ;

-le mot long *n* (en complément à 2, entre -2147483648 et +2147483647) aux adresses *adr*, *adr*+1, *adr*+2 et *adr*+3. L'adresse *adr* doit obligatoirement être paire. Le paramètre *n* peut être négatif, il est alors traité en complément à deux. Voir également les instructions **BYTE**{}, **CARD**{}, **INT**{}, **LONG**{}, {}, **SPOKE**, **SDPOKE** et **SLPOKE**.

SINGLE{*adr*}

adr : mot long.

Cette instruction écrit un nombre en virgule flottante au format IEEE (sur quatre octets) à partir de l'adresse *adr*.

SINGLE{*adr*}

adr : mot long.

Cette fonction lit un nombre en virgule flottante au format IEEE (sur quatre octets) à partir de l'adresse *adr*.

SPOKE *adr*,*n*

SDPOKE *adr*,*n*

SLPOKE *adr*,*n*

adr : mot long ;

n : expression numérique entière.

Ces instructions sont identiques à **POKE**, **DPOKE**, et **LPOKE**, mais en mode superviseur on peut ainsi accéder aux zones-mémoires protégées.

TYPE(*pointeur*)

pointeur : mot long.

Cette fonction renvoie le type de la variable pointée. Ne sert que si l'on travaille avec des adresses (variables précédées par une étoile). Le type renvoyé est pris parmi les suivants :

0 = variable réelle (Float), *variable*# ;

1 = variable chaîne (String), *variable*\$;

2 = variable entière (Integer), *variable*% ;

3 = variable booléenne (Boolean), *variable*! ;

4 = tableau de variables réelles (Float Array), *tableau*#() ;

5 = tableau de variables chaînes (String Array), *tableau*\$() ;

6 = tableau de variables entières (Integer Array), *tableau*%() ;

7 = tableau de variables booléennes (Boolean Array), *tableau*!() ;

8 = variable mot (Word), *variable*& ;

9 = variable octet (Byte), *variable*! ;

12 = tableau de mots (Word Array), *tableau*&() ;

13 = tableau d'octets (Byte Array), *tableau*!() ;

VARPTR(var)

V:var

var : variable.

Cette fonction renvoie l'adresse d'une variable d'un type quelconque, sauf de type tableau. L'adresse d'un élément d'un tableau peut cependant être obtenue. Voir également la fonction **ARRPTR**, et *.

Par exemple :

V:a(12,56) renvoie l'adresse de l'élément d'indices 12 et 56 du tableau a().

FONCTIONS NUMERIQUES

Dans ce chapitre, nous distinguons deux types de fonctions, celle traitant soit une expression ou une variable en virgule flottante que nous avons rebaptisé réelle, soit une expression ou une variable entière. Les dénominations précédemment citées comportent un abus de langage au sens mathématique du terme, toutefois, cela correspond à la meilleure représentation que l'on ait des entiers et des réels.

FONCTIONS DE VARIABLES ENTIÈRES

Le GFA Basic 3.0 comporte une véritable arithmétique entière. Le traitement des variables réelles est également effectué, mais avec des différences par rapport aux versions précédentes :

- si la variable de destination est une variable entière, les conversions de types nécessaires sont effectuées avant les calculs. Par exemple : ADD n,2.23 est identique à ADD n,2 lorsque n'est une variable entière ;
- le gain de rapidité n'est réellement appréciable qu'avec les variables entières, INC n s'exécute beaucoup plus rapidement que n=n+1 lorsque n est une variable entière. Toutefois, MUL x,3.45 s'exécute plus rapidement que x=x*3.45, même lorsque x est une variable réelle ;
- les conversions sont toujours automatiques, mais prenez l'habitude d'utiliser les types appropriés et d'effectuer explicitement les conversions (par une simple affectation), cela afin de s'habituer à une structuration souhaitable et présente dans de nombreux langages (C ou Pascal entre autres).

Voici les instructions et les fonctions traitées dans cette partie :

ADD	DIV()	MUL()
ADD()	EVEN()	ODD()
BIN\$()	FALSE	SUB
CHR\$()	INC	SUB()
DEC	MOD	TRUE
DIV	MUL	

ADD v,n

v : variable numérique ;

n : expression numérique.

Cette fonction additionne l'expression à la valeur de la variable.

Par exemple :

ADD v,2 additionne 2 à la variable v et stocke le résultat dans la variable v. L'instruction ADD v,n est identique à l'affectation n=v+n, mais elle est sensiblement plus rapide (surtout lorsque les calculs s'effectuent avec des entiers).

ADD(n1,n2)

n1 : expression numérique ;

n2 : expression numérique.

Cette fonction renvoie la valeur de la somme de l'expression n1 à l'expression n2.

BIN\$(n)

n : expression numérique entière.

Cette fonction transforme la valeur n en la chaîne de caractères correspondant à la valeur binaire de n.

Par exemple :

PRINT BIN\$(49) affiche 110001.

CHR\$(n)

n : expression numérique entière.

Cette fonction renvoie le caractère dont le code ASCII est n (compris entre 0 et 255).

Par exemple :

PRINT CHR\$(66) affiche B.

Voir également la fonction **ASC**.

DEC n

n : variable numérique.

Cette instruction décrémente la variable de 1. Cette instruction est identique à SUB n,1 ou à n=n-1, mais elle est sensiblement plus rapide (surtout lorsque les calculs s'effectuent avec des entiers).

Voir également l'instruction **INC**.

DIV v,n

v : variable numérique ;

n : expression numérique.

Cette instruction divise la valeur de la variable par l'expression.

Par exemple :

DIV v,2 divise par 2 la variable v et stocke le résultat dans la variable v. L'instruction DIV v,n est identique à l'affectation $v=v/n$, mais elle est sensiblement plus rapide (surtout lorsque les calculs s'effectuent avec des entiers).

DIV(n1,n2)

n1 : expression numérique ;

n2 : expression numérique.

Cette fonction divise l'expression n1 par l'expression n2.

EVEN(n)

n : expression numérique.

Cette fonction détermine si la variable "n" est paire ou impaire. Si elle est paire, la valeur -1 est renvoyée, sinon EVEN retourne 0.

Par exemple :

PRINT EVEN(23) affiche 0.

PRINT EVEN(48) affiche -1.

Voir également la fonction **ODD**.

FALSE

Valeur booléenne "faux", équivalente à 0.

Voir également **TRUE**.

INC n

n : expression numérique.

Cette instruction incrémente la variable de 1. Cette instruction est identique à **ADD n,1** ou à **n=n+1**, mais elle est sensiblement plus rapide (surtout lorsque les calculs s'effectuent avec des entiers).

Voir également l'instruction **DEC**.

MOD v,n

v : variable numérique ;

n : expression numérique.

Cette instruction divise la valeur de la variable par l'expression et affecte le reste à la variable v.

Par exemple :

MOD v,2 divise par 2 la variable v et stocke le reste de la division entière dans la variable v.

MOD(n1,n2) ou n1 MOD n2

n1 : expression numérique ;

n2 : expression numérique.

Cette fonction renvoie le reste de la division entière de l'expression n1 par l'expression n2.

MUL v,n

v : variable numérique ;

n : expression numérique.

Cette instruction multiplie la valeur de la variable par l'expression.

Par exemple :

MUL v,n multiplie la variable v par la valeur de l'expression n et stocke le résultat dans la variable v. L'instruction **MUL v,n** est identique à l'affectation **v=v*n**, mais elle est sensiblement plus rapide (surtout lorsque les calculs s'effectuent avec des entiers).

MUL(n1,n2)

n1 : expression numérique ;

n2 : expression numérique.

Cette fonction multiplie les deux expressions entre elles.

ODD(n)

Cette fonction retourne -1 si n est impair et 0 si n est pair.

Par exemple :

PRINT ODD(-56) affiche 0 .

PRINT ODD(-189) affiche -1 .

voir également la fonction **EVEN**.**SUB v,n**

v : variable numérique ;

n : expression numérique.

Cette instruction soustrait n de la valeur de la variable v.

Par exemple :

SUB v,2 soustrait 2 de la variable v et stocke le résultat dans la variable v.

L'instruction SUB v,n est identique à l'affectation $v=v-n$, mais elle est sensiblement plus rapide (surtout lorsque les calculs s'effectuent avec des entiers).**SUB(n1,n2)**

n1 : expression numérique ;

n2 : expression numérique.

Cette fonction soustrait l'expression n1 de l'expression n2.

TRUE

Valeur booléenne "vrai", équivalent à -1.

INSTRUCTIONS BINAIRES

Voici les fonctions et instructions traitées dans cette partie :

AND	NOT	SHL
BCHG	OR	SHL
BCLR	ROL&	SHR&
BSET	ROL	SHR
BTST	ROL	SHR
BYTE	ROR&	SWAP
CARD	ROR	WORD
EQV	ROR	XOR
IMP	SHL&	

Fonctions logiques

AND(n1,n2)

n1,n2 : expressions numériques entières.

Réalisation d'un ET logique entre n1 et n2.

n1	n2	AND(n1,n2)
1	1	1
1	0	0
0	1	0
0	0	0

EQV(n1,n2)

n1,n2 : expressions numériques entières.

Réalisation de l'équivalence logique entre n1 et n2. Dans le résultat, les bits sont mis à 1 si les bits de même rang sont identiques dans n1 et n2.

n1	n2	EQV(n1,n2)
1	1	1
1	0	0
0	1	0
0	0	1

IMP(n1,n2)

n1,n2 : expressions numériques entières.

Réalisation de l'implication logique entre n1 et n2. Dans le résultat, les seuls bits mis à 0 correspondent aux bits de même rang à 1 dans n1 et à 0 dans n2.

n1	n2	IMP(n1,n2)
1	1	1
1	0	0
0	1	1
0	0	1

NOT(n)

n : expression numérique entière.

Réalisation de la complémentation à 1 de n. Le résultat a tous les bits inversés par rapport à ceux de n.

n	NOT n
1	0
0	1

OR(n1,n2)

n1,n2 : expressions numériques entières.

Réalisation d'un OU logique entre n1 et n2.

n1	n2	OR(n1,n2)
1	1	1
1	0	1
0	1	1
0	0	0

XOR(n1,n2)

n1,n2 : expressions numériques entières.

Réalisation d'un OU exclusif logique entre n1 et n2.

n1	n2	XOR(n1,n2)
1	1	0
1	0	1
0	1	1
0	0	0

Fonctions de décalages et de rotations**ROL(n1,nb)****ROL&(n1,nb)****ROL|(n1,nb)**

n1,nb : expressions numériques entières.

Rotation de n1 vers la gauche. On doit spécifier le nombre nb de rotations élémentaires d'un bit. Contrairement aux décalages, les bits sortis sont réintroduits. Les rotations se font sur un mot long (aucun suffixe), sur un mot (suffixe &, extension du bit 15 aux bits 16 à 31), ou sur un octet (suffixe l, annulation des bits 8 à 31).

ROR(n1,nb)**ROR&(n1,nb)****ROR|(n1,nb)**

n1,nb : expressions numériques entières.

Rotation de n1 vers la droite. On doit spécifier le nombre nb de rotations élémentaires d'un bit. Contrairement aux décalages, les bits sortis sont réintroduits. Les rotations se font sur un mot long (aucun suffixe), sur un mot (suffixe &, extension du bit 15 aux bits 16 à 31), ou sur un octet (suffixe l, annulation des bits 8 à 31).

SHL(n1,nb)**SHL&(n1,nb)****SHL|(n1,nb)**

n1,nb : expressions numériques entières.

Décalage de n1 vers la gauche. On doit spécifier le nombre nb de décalages élémentaires d'un bit. Les décalages se font sur un mot long (aucun suffixe), sur un mot (suffixe &, extension du bit 15 aux bits 16 à 31), ou sur un octet (suffixe l, annulation des bits 8 à 31).

SHR(n1,nb)**SHR&(n1,nb)****SHR|(n1,nb)**

n1,nb : expressions numériques entières.

Décalage de n1 vers la droite. On doit spécifier le nombre nb de décalages élémentaires d'un bit. Les décalages se font sur un mot long (aucun suffixe), sur un mot (suffixe &, extension du bit 15 aux bits 16 à 31), ou sur un octet (suffixe l, annulation des bits 8 à 31).

Fonctions de manipulations de bits

BCHG(n,i)

n,i : expressions numériques entières.

Cette fonction positionne le i^e bit de n dans son état complémentaire (mise à 1 s'il était à 0 et mise à 0 s'il était à 1) et renvoie la valeur obtenue.

BCLR(n,i)

n,i : expressions numériques entières.

Cette fonction positionne le i^e bit de n à 0 et renvoie la valeur obtenue.

BSET(n,i)

n,i : expressions numériques entières.

Cette fonction positionne le i^eème bit de n à 1 et renvoie la valeur obtenue.

BTST(n,i)

n,i : expressions numériques entières.

Cette fonction teste le i^e bit de n. BTST renvoie une valeur booléenne : TRUE si le bit considéré est à 1, FALSE si ce bit est à 0. Le paramètre i doit être compris entre 0 et 31 (il est de toute façon réajusté dans l'intervalle 0-31 par l'interpréteur, avec un AND 31).

BYTE(n)

n : expression numérique entière.

Cette fonction renvoie l'octet poids faible (ou inférieur) de l'expression entière n.

CARD(n)

n : expression numérique entière.

Cette fonction renvoie le mot poids faible (ou inférieur) de l'expression entière n.

SWAP(n)

n : mot long.

Cette instruction permute les mots poids fort et poids faible d'un mot long, c'est-à-dire que SWAP effectue l'échange des 16 bits supérieurs et des 16 bits inférieurs d'un mot long.

WORD(n)

n : expression numérique entière.

Cette instruction effectue l'extension d'un mot à un mot long : le bit de signe (bit 15) du mot est recopié dans les bits 16 à 31. Cette opération assure la conversion entre mots signés et mots longs signés.

FONCTIONS DE VARIABLES REELLES

Voici les fonctions et instructions traitées dans cette partie :

ABS	LOG10	SIN
ACOS	MAX	SINQ
ASIN	MIN	SQR
ATN	PI	STRS
COS	PRED	SUCC
COSQ	RAD	TAN
DEG	RAND	TRUNC
EXP	RANDOM	
FIX	RANDOMIZE	
FRAC	RND	
INT	ROUND	
LOG	SGN	

ABS(x)

x : expression numérique.

Cette fonction renvoie la valeur absolue de x (sans le signe).

Par exemple :

PRINT ABS(-1) affiche 1.

ACOS(x)

x : expression numérique.

Cette fonction calcule l'arccosinus de x, avec x exprimé en radians.

Voir également les fonctions ASIN et ATN.

ASIN(x)

x : expression numérique.

Cette fonction calcule l'arcsinus de x, avec x exprimé en radians.

Voir également les fonctions ACOS et ATN.

ATN(x)

x : expression numérique.

Cette fonction calcule l'arctangente de x, avec x exprimé en radians.

Voir également les fonctions ACOS et ASIN.

COS(x)

x : expression numérique.

Cette fonction calcule le cosinus de x, avec x exprimé en radians.

Voir également les fonctions SIN et TAN.

COSQ(x)

x : expression numérique.

Cette fonction calcule le cosinus de x de manière par interpolation avec une table de sinus par intervalles de degrés interne au GFA 3.0. Le paramètre "x" est exprimé en degrés.

Voir également la fonction **SINQ**.

DEG(x)

x : expression numérique.

Cette fonction convertit un angle exprimé en radians en un angle exprimé en degrés. Elle est particulièrement utilisée pour les fonctions **COSQ** et **SINQ**.

Voir également la fonction **RAD**.

EXP(x)

x : expression numérique.

Cette fonction calcule l'exponentielle de x. Fonction inverse du logarithme népérien.

Voir également la fonction **LOG**.

FIX(x)

x : expression numérique.

Cette fonction retourne la partie entière de x, c'est-à-dire celle située avant la virgule. **FIX** est équivalent à **TRUNC**.

Par exemple :

PRINT FIX(-36.789) affiche -36.

Voir également les fonctions **FRAC**, **INT**, **ROUND** et **TRUNC**.

FRAC(x)

x : expression numérique.

Cette fonction retourne la partie décimale de x, c'est-à-dire celle située après la virgule.

Par exemple :

PRINT FRAC(-36.789) affiche -0.789.

Voir également les fonctions **FIX**, **INT**, **ROUND** et **TRUNC**.

INT(x)

x : expression numérique.

Cette fonction calcule la partie entière de x. Si x est positif, la partie décimale est tronquée, sinon on arrondit x dans le sens des x négatifs. Attention, **INT** n'est pas équivalente à **TRUNC** pour x négatif!

Par exemple :

PRINT INT(-36.789) affiche -37.

Voir également les fonctions **FIX**, **FRAC**, **ROUND** et **TRUNC**.

LOG(x)

x : expression numérique.

Cette fonction calcule le logarithme népérien de x.

Voir également la fonction **EXP**.

LOG10(x)

x : expression numérique.

Cette fonction calcule le logarithme décimal de x.

Voir également la fonction **LOG**.

MAX(expr1[,expr2,...])

expr1,expr2 : expressions numériques.

Cette fonction renvoie l'expression dont la valeur est la plus grande.

Par exemple :

PRINT MAX(-56,-12,3) affiche 3.

Voir également la fonction **MIN**.

MIN(expr1[,expr2,...])

expr1,expr2 : expressions numériques.

Cette fonction renvoie l'expression dont la valeur est la plus petite.

Par exemple :

PRINT MIN(-56,-12,3) affiche -56.

Voir également la fonction **MAX**.

PI

Valeur de PI, 3.145926536...

PRED(n)

n : expression numérique entière.

Cette fonction renvoie l'entier qui succède à n.

Par exemple :

n=5

PRINT PRED(n)

affiche 4.

Voir également la fonction **SUCC**.

RAD(x)

x : expression numérique.

Cette fonction convertit un angle exprimé en degrés en un angle exprimé en radians.

Voir également la fonction **DEG**.

RAND(x)

x : expression numérique.

Cette fonction renvoie un nombre aléatoire entier (sur un mot) compris entre 0 inclus et x exclu.

Voir également l'instruction **RANDOMIZE** et les fonctions **RANDOM** et **RND**.

RANDOM(x)

x : expression numérique entière.

Cette fonction renvoie un nombre aléatoire compris entre 0 inclus et x exclu.

Voir également l'instruction **RANDOMIZE** et les fonctions **RAND** et **RND**.

RANDOMIZE [x]

x : expression numérique.

Cette instruction initialise le générateur de nombres aléatoires. Si le paramètre "x" est précisé, la séquence de nombres générés sera fonction du nombre "x".

Voir également l'instruction **RAND** et les fonctions **RANDOM** et **RND**.

RND[(x)]

x : expression numérique.

Cette fonction renvoie un nombre aléatoire compris entre 0 inclus et 1 exclu. Le paramètre x est facultatif, et même s'il est précisé, il n'est pas pris en compte.

Voir également l'instruction **RANDOMIZE** et les fonctions **RAND** et **RANDOM**.

ROUND(x[,n])

x : expression numérique ;

n : expression numérique entière.

Cette fonction renvoie la valeur arrondie de l'expression x. Le paramètre "n" indique le nombre de chiffres après la virgule à conserver lors de l'arrondi. Si n est nul, **ROUND** retourne un nombre entier, si n est négatif, l'arrondi se fait à la dizaine supérieure avant la virgule.

Par exemple :

ROUND(12,2) affiche 13 ;

ROUND(-12,2) affiche -12 ;

ROUND(12,-1) affiche 20.

Voir également les fonctions **FIX**, **FRAC**, **INT** et **TRUNC**.

SGN(x)

x : expression numérique.

Cette fonction détermine si l'expression x est positive, négative ou nulle (1, -1 ou 0).

Par exemple :

PRINT SGN(-5693.98) affiche -1;

PRINT SGN(5693.98) affiche 1;

PRINT SGN(0) affiche 0.

SIN(x)

x : expression numérique.

Cette fonction calcule le sinus de x, avec x exprimé en radians.

Voir également les fonctions **COS** et **TAN**.

SINQ(x)

x : expression numérique.

Cette fonction calcule le sinus de x de manière par interpolation avec une table de sinus par intervalles de degrés interne au GFA 3.0. Le paramètre "x" est exprimé en degrés.

Voir également la fonction **COSQ**.

SQR(x)

x : expression numérique.

Cette fonction calcule la racine carrée de x.

STR\$(x)

x : expression numérique.

Transforme une valeur numérique x en la chaîne de caractères correspondant à la valeur en base 10 (décimal) de x.

SUCC(n)

n : expression numérique entière.

Cette fonction renvoie l'entier qui succède à n.

Par exemple :

n=5

PRINT SUCC(n)

affiche 6.

Voir également la fonction **PRED**.

TAN(x)

x : expression numérique.

Cette fonction calcule la tangente de x, avec x exprimé en radians.

Voir également les fonctions **COS** et **SIN**.

TRUNC(x)

x : expression numérique.

Cette fonction renvoie la partie entière de x en supprimant purement et simplement sa partie décimale. **TRUNC** est équivalent à **FIX**.

Par exemple :

PRINT TRUNC(-36.789) affiche -36.

Voir également les fonctions **FIX**, **FRAC**, **INT** et **ROUND**.

TRAITEMENT DES CHAINES DE CARACTERES

Ce chapitre est dédié à toutes les fonctions destinées au traitement des expressions alphanumériques, à l'exception des opérations très particulières concernant leur manipulation directe en mémoire, la gestion des tableaux (tri, insertion et suppression d'éléments...) ou les conversions.

Une chaîne alphanumérique est délimitée par une paire de guillemets (encore appelés double quotes). Pour introduire un guillemet dans une chaîne, il suffit de le répéter deux fois, mais il n'apparaîtra qu'une seule fois. L'affectation se fait avec l'opérateur = et la concaténation avec l'opérateur +.

Par exemple :

```
ch1$="un guillemet "" se trouve tout seul,"
```

```
ch2$=" en voici deux : """" !" "
```

```
ch$=ch1$+ch2$
```

```
PRINT ch$
```

affiche

un guillemet " se trouve tout seul, en voici deux : "" !

Voici les instructions et les fonctions traitées dans ce chapitre :

ASC	MID\$	SPACE\$
INSTR	MIN	SPC
LEFT\$	PRED	STRING\$
LEN	RIGHT\$	SUCC
LSET	RINSTR	TRIM\$
MAX	RSET	UPPER\$

ASC(ch\$)

ch\$: expression alphanumérique.

Cette fonction renvoie le code ASCII du premier caractère de la chaîne, ou 0 si cette dernière est vide.

Par exemple :

```
PRINT ASC("Table") affiche 84.
```

Voir également les fonctions **CHR\$** et **STR\$**.

INSTR([n,]ch1\$,ch2\$)

INSTR(ch1\$,ch2\$[,n])

ch1\$,ch2\$: expression alphanumérique ;

n : expression numérique entière.

Cette fonction recherche la chaîne de caractères ch2\$ dans la chaîne ch1\$ en commençant par la droite. Si ch1\$ contient ch2\$, **INSTR** renvoie la position de celle-ci, sinon la valeur 0 est retournée. Le paramètre facultatif "n" permet de commencer la recherche à partir de la position n.

Par exemple :

```
ch$="traitement des chaînes de caractères"
```

PRINT INSTR(ch\$,"ai",5)

affiche 18.

Voir également la fonction **RINSTR**.

LEFT\$(ch\$,n)

ch\$: expression alphanumérique ;

n : expression numérique entière.

Cette fonction renvoie le premier caractère de la chaîne ch\$. Si le paramètre optionnel "n" est indiqué, les n premiers caractères de la chaîne sont renvoyés. Si n est supérieur à la longueur de la chaîne, celle-ci est renvoyée dans son intégralité.

Par exemple :

ch\$="traitement des chaînes de caractères"

PRINT LEFT\$(ch\$,10)

affiche traitement.

Voir également les fonctions **MID\$** et **RIGHT\$**.

LEN(ch\$)

ch\$: expression alphanumérique.

Cette fonction renvoie la longueur de la chaîne de caractères.

Par exemple :

ch\$="traitement des chaînes de caractères"

PRINT LEN(ch\$)

affiche 36.

LSET ch1\$=ch2\$

ch1\$: variable alphanumérique ;

ch2\$: expression alphanumérique.

Cette instruction affecte la chaîne ch2\$ à la chaîne ch1\$ en justifiant à gauche. La chaîne ch1\$ doit être préalablement déclarée. Son contenu originel n'a aucune importance, seule compte sa longueur. Si la chaîne ch2\$ comporte un nombre de caractères supérieur à celui de ch1\$, l'insertion s'interrompt dès que ch1\$ est remplie. Dans le cas contraire, l'espace restant est complété de blancs.

Par exemple :

ch\$=SPACE\$(25)

LSET ch\$="justification à gauche"

PRINT "*" ; ch\$; "*"

affiche *justification à gauche *.

Voir également l'instruction **RSET**.

MAX ch1\$ [,ch2\$,...]

ch1\$: expression alphanumérique ;

ch2\$: expression alphanumérique.

Cette fonction renvoie la plus grande des deux (ou plus) chaînes de caractères.

Voir également la fonction **MIN**.

MID\$(ch\$,i[,n])

ch\$: expression alphanumérique ;

i,n : expression numérique entière.

Cette fonction renvoie la sous-chaîne de ch\$ comportant n caractères à partir de la ième position.

Par exemple :

ch\$="traitement des chaînes de caractères"

PRINT MID\$(ch\$,13,2)

affiche es.

Voir également les fonctions **LEFT\$** et **RIGHT\$**.

MID\$(ch1\$,i[,n])=ch2\$

ch1\$: variable alphanumérique ;

ch2\$: expression alphanumérique ;

i,n : expression numérique entière.

Cette instruction remplace la sous-chaîne de ch1\$ commençant au i^e caractère par n caractères de la chaîne ch2\$ (ou la totalité de ch2\$ si n est absent). La longueur de ch1\$ n'est pas modifiée, les caractères supplémentaires sont tronqués.

Par exemple :

ch\$="traitement des chaînes de caractères"

MID\$(ch\$,3,4)="aitement "

PRINT ch\$

affiche traitement des chaînes de caractères.

MIN ch1\$ [,ch2\$,...]

ch1\$: expression alphanumérique ;

ch2\$: expression alphanumérique.

Cette fonction renvoie la plus petite des deux (ou plus) chaînes de caractères.

Voir également la fonction **MAX**.

PRED(ch\$)

ch\$: expression alphanumérique ;

Cette fonction renvoie le caractère qui précède, dans la table ASCII, le premier caractère de ch\$.

Par exemple :

ch\$="table"

PRINT PRED(ch\$)

affiche s.

Voir également la fonction **SUCC**.

RIGHT\$(ch\$,n)

ch\$: expression alphanumérique ;

n : expression numérique entière.

Cette fonction retourne le dernier caractère de la chaîne ch\$. Si le paramètre optionnel n est indiqué, les n derniers caractères de la chaîne sont renvoyés. Si n est supérieur à la longueur de la chaîne, celle-ci est renvoyée dans son intégralité.

Par exemple :

ch\$="traitement des chaînes de caractères"

PRINT RIGHT\$(ch\$,10)

affiche caractères.

Voir également les fonctions **LEFT\$** et **MID\$**.

RINSTR([n],ch1\$,ch2\$)

RINSTR(ch1\$,ch2\$[,n])

ch1\$,ch2\$: expression alphanumérique ;

n : expression numérique entière.

Cette fonction recherche la chaîne ch2\$ dans la chaîne ch1\$ en commençant par la gauche. Si ch1\$ contient ch2\$, **RINSTR** renvoie la position de celle-ci, sinon la valeur 0 est retournée. Le paramètre facultatif "n" permet de commencer la recherche à partir de la position (n-1). En effet, lorsque vous indiquez ce paramètre, il faut rajouter 1 à la position désirée pour obtenir un résultat correct, même si cela dépasse la longueur de la chaîne (petite erreur du langage).

Par exemple :

ch\$=" traitement des chaînes de caractères "

PRINT RINSTR(ch\$,"es",LEN(ch\$)+1)

affiche 36.

Voir également la fonction **INSTR**.

RSET ch1\$=ch2\$

ch1\$: variable alphanumérique ;

ch2\$: expression alphanumérique.

Cette instruction affecte la chaîne ch2\$ à la chaîne ch1\$ en justifiant à droite. La chaîne ch1\$ doit être préalablement déclarée, son contenu originel n'a aucune importance, seule compte sa longueur. Si la chaîne ch2\$ comporte un nombre de caractères supérieur à celui de ch1\$, l'insertion s'interrompt dès que ch1\$ est remplie. Dans le cas contraire, l'espace restant est complété de blancs.

Par exemple :

ch\$=SPACE\$(25)

RSET ch\$="justification à droite"

PRINT "*" ;ch\$;"+"

affiche * justification à droite*.

SPACE\$(n)

n : expression numérique entière.

Cette instruction définit une chaîne de caractères composée de n blancs.

Voir également les instructions **SPC** et **STRING\$**.

SPC(n)

n : expression numérique entière.

Cette instruction, conjointement utilisée avec l'instruction **PRINT**, affiche n caractères blancs (espaces). Mais contrairement à **SPACE\$**, il n'est pas possible de l'utiliser dans les expressions alphanumériques.

Par exemple :

```
ch$=SPACE$(3)
```

```
PRINT "bonjour";ch$;"à";SPC(5);"tous"
```

affiche bonjour à tous.

Voir également les instructions **SPACE\$** et **STRING\$**.

STRING\$(n,ch\$)**STRING\$(n,code)**

n,code : expression numérique entière ;

ch\$: expression alphanumérique.

Cette instruction définit une chaîne de caractères de n fois la chaîne ou le caractère dont le code ASCII est indiqué.

Par exemple :

```
ch$="ha"
```

```
PRINT STRING$(3,ch$)
```

affiche hahaha.

Voir également les instructions **SPACE\$** et **SPC**.

SUCC(ch\$)

ch\$: expression alphanumérique.

Cette fonction renvoie le caractère qui succède, dans la table ASCII, au premier caractère de ch\$.

Par exemple :

```
ch$="table"
```

```
PRINT SUCC(ch$)
```

affiche u. Voir également la fonction **PRED**.

TRIM\$(ch\$)

ch\$: expression alphanumérique.

Cette fonction renvoie une chaîne constituée de la chaîne ch\$ privée de tous les espaces de début et de fin de chaîne.

Par exemple :

```
ch$="   espace   "
PRINT "*" ; TRIM$(ch$) ; "*"
affiche *espace*
```

UPPER\$(ch\$)

ch\$: expression alphanumérique.

Cette instruction transforme toutes les lettres minuscules de la chaîne en majuscules (même les caractères accentués).

Par exemple :

```
PRINT UPPER$("MajusCuLeS") affiche MAJUSCULES.
```

EXEMPLES DE TRAITEMENT DES CHAINES DE CARACTERES

Chaque programme est donné dans son intégralité et décortiqué afin de vous expliquer comment fonctionne les diverses instructions. Les programmes fonctionnent dans la mesure du possible dans les trois résolutions de l'Atari ST.

PALINDROMES ET ANAGRAMMES

75

Ce programme se divise en deux parties distinctes : la recherche d'anagrammes et la reconnaissance de palindromes. Un palindrome est un mot qui se lit dans les deux sens. Par exemple, ARA, ICI, LAVAL, ESOPE...

Programme principal

Les instructions suivantes sont facultatives, toutefois, elles faciliteront l'écriture du programme. En effet, lorsque les variables sont saisies (mode DEFLIST 2), elles prennent automatiquement le bon suffixe à l'affichage.

DEFBYT "b" désigne des octets ;
DEFWRD "l,p" désigne des mots (16 bits) ;
DEFSTR "mot,car" désigne des chaînes.

Variable globale :

b| réponse à la boîte de dialogue.

Procédure palindrome

Cette procédure teste si un mot est un palindrome, c'est-à-dire si ce mot est égal à son inverse.

Variable locale :

mot\$ contient le mot à tester.

Procédure anagramme

Cette procédure teste si deux mots sont des anagrammes, c'est-à-dire si les deux mots sont constitués des mêmes lettres arrangées différemment. Pour ce faire, on recopie les deux mots dans des variables temporaires, puis, on teste l'existence de chaque lettre du premier mot dans le second. Enfin, la lettre est enlevée de chacun des mots provisoires, afin de traiter les cas des lettres présentes plusieurs fois.

Variables locales :

mot1\$ contient le premier mot ;
mot2\$ contient le second mot ;
ana1\$ contient le premier mot provisoire ;
ana2\$ contient le second mot provisoire ;
car\$ contient la lettre de test ;
ana! contient le résultat des tests.

Fonction inverse_mot\$

Cette fonction renvoie un mot inversé de manière récursive.

Paramètre :

mot\$ contient le mot à inverser.

Variable locale :

l& longueur du mot à inverser.

Programme :

```

' -----
'   PALINDROME ET ANAGRAMME
' -----
'
' DEFBYT "b"
' DEFWRD "l,p"
' DEFSTR "'mot,car"
'
REPEAT
  ALERT 2,"A:ANAGRAMME|B:PALINDROME",RAND(2)+1,"A|B",b|
  ON b| GOSUB anagramme,palindrome
  ALERT 2,"FIN",1,"OUI|NON",b|
UNTIL b|=1
END
' -----
'   PROCEDURES ET FONCTIONS
' -----

PROCEDURE palindrome
  LOCAL mot$
  CLS
  PRINT
  AT(30,2);CHR$(27)+"p";"PALINDROME";CHR$(27)+"q";CHR$(10)
  PRINT " Donnez un mot !"
  PRINT
  INPUT " Votre mot ? ",mot$
  PRINT
  IF mot$=@inverse_mot$(mot$) THEN
    PRINT mot$;" est un palindrome"
  ELSE
    PRINT mot$;" n'est pas un palindrome"
  ENDIF
  RETURN
'
PROCEDURE anagramme
  LOCAL mot1$,mot2$,ana1$,ana2$,car$,ana!
  CLS
  PRINT

```

```

AT(30,2);CHR$(27)+"p";"ANAGRAMME";CHR$(27)+"q";CHR$(10)
INPUT "premier mot ? ",mot1$
PRINT
INPUT "second mot ? ",mot2$
PRINT
ana1$=mot1$
ana2$=mot2$
ana!=TRUE
IF LEN(mot1$)=LEN(mot2$) THEN
  REPEAT
    car$=LEFT$(ana1$,1)
    p&=INSTR(ana2$,car$)
    IF p&=0 THEN
      ana!=FALSE
    ELSE
      ana1$=MID$(ana1$,2)
      ana2$=LEFT$(ana2$,p&-1)+MID$(ana2$,p&+1)
    ENDIF
  UNTIL ana1$="" OR ana!=FALSE
ELSE
  ana!=FALSE
ENDIF
IF ana! THEN
  PRINT mot1$;" et ";mot2$;" sont deux anagrammes"
ELSE
  PRINT mot1$;" et ";mot2$;" ne sont pas deux anagrammes"
ENDIF
RETURN
,
FUNCTION inverse_mot$(mot$)
  LOCAL l&
  l&=LEN(mot$)
  IF l&>1 THEN
    RETURN RIGHT$(mot$,1)+@inverse_mot$(LEFT$(mot$,l&-1))
  ELSE
    RETURN mot$
  ENDIF
ENDFUNC

```

ANAGRAMME

premier mot ? M. PAIRS
 second mot ? M. PARIS
 M. PAIRS et M. PARIS sont deux anagrammes

PALINDROME

Donnez un mot !
 Votre mot ? LAVAL
 LAVAL est un palindrome

NOTATION POLONAISE INVERSE

Ce programme transforme une notation algébrique normale (infixée) en une notation postfixée (polonaise inverse).

Programme principal

Les instructions suivantes sont facultatives, toutefois, elles faciliteront l'écriture du programme. En effet, lorsque les variables sont saisies (mode DEFLIST 2), elles prennent automatiquement le bon suffixe à l'affichage.

DEFBYT "i,n,l,p,p_loc" désigne des octets ;
 DEFSTR "ch,p_" désigne des chaînes ;
 DEFSTR "exp,car" désigne des chaînes.

Variables globales :

n| taille des piles ;
 p_post\$() pile postfixée ;
 p_op\$() pile des opérateurs ;
 inf_exp\$ expression infixée à convertir en notation postfixée.

Procédure empile

Cette procédure empile dans la pile considérée la variable alphanumérique ch\$.

Paramètres :

ch\$ variable à empiler ;
 pile\$() pile utilisée.

Variable locale :

p| pointeur local de pile.

Fonction depile\$

Cette fonction renvoie le sommet de la pile considérée.

Paramètre :

pile\$() pile utilisée.

Variable locale :

p| pointeur local de pile.

Procédure voir_pile

Cette procédure affiche la pile spécifiée.

Paramètre :

pile\$() pile utilisée.

Variables locales :

i| indice de boucle ;

p| pointeur local de pile.

Fonction polonaise

Cette fonction récursive transforme la notation infixée en notation postfixée.

Paramètre :

inf_exp\$ expression infixée ;

Variables locales :

long| longueur de l'expression infixée ;
p_car| pointeur de caractères ;
p_loc| pointeur de pile local ;
car\$ caractère courant ;
exp\$ expression en cours d'élaboration.

Procédure empile_exp

Cette procédure empile dans la pile postfixée (p_post\$()) l'expression exp\$, si celle-ci n'est pas nulle.

Fonction priorité

Cette fonction renvoie la priorité de l'opérateur considéré.

Paramètre :

op\$ opérateur.

Variable locale :

prio| priorité de l'opérateur.

Programme :

```

' -----
'   NOTATION POLONAISE
' -----
'
' DEFBYT "i,n,l,p,p_loc"
' DEFSTR "ch,p_"
' DEFSTR "exp,car"
'
n|=200
DIM p_post$(n|)
DIM p_op$(n|)
p_post$(0)=CHR$(1)
p_op$(0)=CHR$(1)
inf_exp$="( (8*COS(45)-
LOG(78/3)+EXP((2*3)+TAN((3+2))*PI))+COS(SIN(3*45^2))) "
~@polonaise(inf_exp$)
PRINT "Votre expression : "
PRINT
PRINT inf_exp$
PRINT
PRINT "Notation polonaise : "
PRINT
voir_pile(p_post$())
~INP(2)
END
' -----

```

```

'      PROCEDURES ET FONCTIONS
'      -----
PROCEDURE empile(ch$,VAR pile$())
  LOCAL p|
  p|=ASC(pile$(0))
  pile$(p|)=ch$
  INC p|
  pile$(0)=CHR$(p|)
RETURN
'

FUNCTION depile$(VAR pile$())
  LOCAL p|
  p|=ASC(pile$(0))
  DEC p|
  pile$(0)=CHR$(p|)
  IF p|=0 THEN
    INC p|
    pile$(0)=CHR$(p|)
    RETURN ""
  ELSE
    RETURN pile$(p|)
  ENDIF
ENDFUNC
'

PROCEDURE voir_pile(VAR pile$())
  LOCAL i|,p|
  p|=ASC(pile$(0))
  FOR i|=1 TO p|
    PRINT pile$(i|)'
  NEXT i|
  PRINT
RETURN
'

FUNCTION polonaise(inf_exp$)
  LOCAL long|,p_car|,p_loc|,car$,exp$
  inf_exp$=UPPER$(inf_exp$)
  long|=LEN(inf_exp$)
  p_car|=0
  p_loc|=0
  exp$=""
  REPEAT
    INC p_car|
    car$=MID$(inf_exp$,p_car|,1)
    SELECT car$
    CASE "+","-","*","/","MOD","DIV","^"
      empile_exp
    IF p_loc|>0 THEN

```

```

exp$=@depile$(p_op$())
IF @priorite(exp$)>=@priorite(car$) THEN
  empile_exp
  empile(car$,p_op$())
ELSE
  empile(exp$,p_op$())
  exp$=""
  empile(car$,p_op$())
  INC p_loc|
ENDIF
ELSE
  empile(car$,p_op$())
  INC p_loc|
ENDIF
CASE ")"
  empile_exp
  WHILE p_loc|>0
    exp$=@depile$(p_op$())
    DEC p_loc|
    empile_exp
  WEND
CASE "("
  SELECT exp$
  CASE "SIN","COS","TAN","EXP","LOG"
    empile(exp$,p_op$())
    ADD p_car|,@polonaise(MID$(inf_exp$,ADD(p_car|,1)))
    exp$=@depile$(p_op$())
    empile_exp
  DEFAULT
    ADD p_car|,@polonaise(MID$(inf_exp$,ADD(p_car|,1)))
  ENDSELECT
DEFAULT
  exp$=exp$+car$
ENDSELECT
UNTIL car$=")" OR p_car|=long|
RETURN p_car|
ENDFUNC
'
PROCEDURE empile_exp
  IF exp$<>"" THEN
    empile(exp$,p_post$())
    exp$=""
  ENDIF
RETURN
'
FUNCTION priorite(op$)
  LOCAL prio|

```

```
SELECT op$
CASE "^"
  prio|=7
CASE "*", "/"
  prio|=6
CASE "MOD", "DIV"
  prio|=5
CASE "+", "-"
  prio|=4
ENDSELECT
RETURN prio|
ENDFUNC
```

Votre expression :

$(\sin(3 * (\cos(\log(5 / (\exp(545)^7)) + \tan(\sin(\alpha + 2))))))$

Notation polonaise :

3 5 545 EXP 7 ^ / LOG ALPHA 2 + SIN TAN + COS * SIN

INSTRUCTIONS DE STRUCTURES

Ce chapitre présente les instructions de structures, c'est-à-dire celles qui permettent d'intervenir sur le déroulement d'un programme et en fin de compte de le faire fonctionner. L'absence de ces instructions entraînerait un déroulement purement linéaire, sans aucune possibilité de décision.

Hormis les instructions de décision, la modularisation est un aspect extrêmement important dans un langage puisqu'elle permet de résoudre des problèmes complexes en les découpant en modules indépendants beaucoup plus faciles à mettre au point. Les procédures et fonctions sont la clef de voûte de ce style de programmation abolissant à tout jamais les inextricables branchement absolus du type GOTO.

Il convient de présenter préalablement la notion de pile et de portée d'une variable, afin de comprendre l'utilisation des procédures et fonctions.

Lors de l'appel d'une procédure ou d'une fonction, tous les paramètres transmis par valeur dans les variables déclarées dans l'en-tête (voir l'instruction VAR pour les autres cas), ainsi que les variables locales, sont placés dans une pile et toute modification est effectuée sur cette zone mémoire. Elles conservent leurs valeurs dans le corps de la procédure ou de la fonction ainsi que dans toutes les procédures ou fonctions appelées (tant qu'elles n'ont pas été redéclarées). Lors du retour au niveau appelant, leurs valeurs seront définitivement perdues. La collision entre deux variables de même nom, l'une interne à la procédure et l'autre externe, est rendue impossible, seule la variable locale sera modifiée. Toutefois, une variable qui n'a pas été déclarée locale ou déclarée dans l'en-tête sera manipulée comme une variable globale et toute modification sera répercutée au niveau appelant. Les variables passées en paramètre avec VAR constituent un cas particulier; bien que déclarées dans l'en-tête, toute modification sera répercutée au niveau appelant. Grâce à cette sauvegarde dans une pile LIFO, on obtient une gestion claire en fonction du niveau où l'on se trouve : à un niveau donné correspond un contexte de variables. Cette facilité prend toute sa puissance avec les procédures ou les fonctions récursives.

Voici les instructions traitées dans ce chapitre :

DEFFN/FN	IF/ELSE/ENDIF
DO/LOOP	LOCAL
DO/UNTIL	ON GOSUB
DO/WHILE	PROCEDURE/RETURN
ELSEIF	REPEAT/UNTIL
EXIT IF	RETURN
FOR/NEXT	SELECT/ENDSELECT
FUNCTION/ENDFUNC	VAR
GOSUB	WHILE/WEND
GOTO	

DEFFN func[(par1,par2...)] = exp

FN func[(par1,par2...)]

exp, par1, par2... : expressions.

Cette instruction définit une fonction utilisateur sur une seule ligne l'expression exp est évaluée, en tenant compte des paramètres facultatifs spécifiés. Ceux-ci sont locaux à la fonction. l'appel s'effectue avec l'instruction FN suivie du nom de la fonction, ainsi que de la liste des paramètres si besoin est.

Voir également l'instruction **FUNCTION**.

DO/LOOP

Cette instruction exécute, en une boucle sans fin, toutes les instructions comprises entre DO et LOOP. Seule l'instruction EXIT permet de sortir de cette boucle ou bien l'appui simultané sur <CONTROL>, <SHIFT> et <ALTER-NATE> qui stoppe le programme.

DO WHILE condition/DO UNTIL condition

LOOP WHILE condition/LOOP UNTIL condition

condition : expression numérique booléenne.

Ces instructions permettent d'avoir une panoplie de boucles avec de multiples possibilités de contrôle. Les instructions DO et LOOP peuvent se voir rajouter les clauses supplémentaires UNTIL et WHILE. Rappelons que UNTIL condition recherche la vérification de la condition pour sortir de la boucle et que WHILE condition provoque la poursuite de la boucle tant que la condition est remplie. On obtient alors neuf boucles possibles (DO/LOOP tout seul y compris).

Par exemple :

DO WHILE condition1

instructions ...

LOOP UNTIL condition2

Cette boucle se poursuivra tant que la première condition sera remplie ou jusqu'à ce que la seconde le soit.

DO UNTIL condition1

instructions ...

LOOP WHILE condition2

Cette boucle se poursuivra tant que la seconde condition sera remplie ou jusqu'à ce que la première le soit.

DO WHILE/LOOP est équivalente à l'instruction **WHILE/WEND**

et **DO/LOOP UNTIL** à l'instruction **REPEAT/UNTIL**.

Voir également les instructions **REPEAT/UNTIL** et **WHILE/WEND**.

ELSEIF condition

condition : expression numérique booléenne.

Cette instruction évite les imbrications de IF/THEN/ELSE qui peuvent vite devenir inextricables. Elle remplace une série d'instructions IF/ELSE/ENDIF interne à une autre structure IF/ELSE/ENDIF. Une structure IF/ELSEIF/ELSE/ENDIF teste tout d'abord la condition après IF. Si cette dernière est

vérifiée, la suite d'instructions entre le IF et le ELSEIF qui le suit est effectuée, puis la boucle se termine avec ENDIF. Sinon la condition après le ELSEIF est testée. Si celle-ci est vérifiée, la séquence d'instructions entre ce ELSEIF et le prochain ELSEIF ou ELSE est exécutée, puis la boucle se termine avec ENDIF, et ainsi de suite... Si aucune condition suivant un ELSEIF n'est vérifiée, la séquence d'instructions entre ELSE et ENDIF est exécutée.

Par exemple :

```
IF a=1 THEN
```

```
...
```

```
ELSEIF a=2 and a$="titi"
```

```
...
```

```
ELSEIF b$="toto"
```

```
...
```

```
ELSE
```

```
...
```

```
ENDIF
```

Si a=1, les instructions situées après le IF sont exécutées; sinon si a=2 et a\$="titi", les instructions situées après ce ELSEIF sont exécutées; sinon si b\$="toto", les instructions situées entre le ELSEIF correspondant et le ELSE s'effectuent; enfin, en dernier recours la séquence entre le ELSE et ENDIF est exécutée.

Voir également l'instruction IF/THEN//ELSE/ENDIF.

EXIT IF condition

condition : expression numérique booléenne.

Cette instruction sort d'une boucle FOR/NEXT, DO/LOOP (ainsi que les combinaisons avec WHILE et UNTIL), WHILE/WEND, ou REPEAT/UNTIL. Attention, si plusieurs boucles s'imbriquent, l'instruction EXIT permet de sortir du niveau de boucle où l'on se trouve! Par exemple :

```
DO
```

```
PRINT
```

```
WHILE A<10000
```

```
INC A
```

```
EXIT IF INKEY$="E"
```

```
WEND
```

```
PRINT
```

```
LOOP
```

La sortie s'effectue au niveau du PRINT précédant le LOOP si l'on appuie sur la touche "E".

FOR i=vdebut [DOWN]TO vfin[STEP pas]/NEXT i

i : variable numérique ;

vdebut,vfin,pas : expression numérique.

Cette instruction exécute, un nombre de fois fixé, toutes les instructions comprises entre FOR et NEXT. Lors de l'exécution du FOR, la valeur vdebut

est affectée à i. NEXT effectue les actions suivantes :

- si le pas est positif; la variable i est incrémentée de ce pas. Si i est inférieur ou égal à la valeur vfin, l'exécution se poursuit à l'instruction suivant FOR et on itère encore une fois, sinon on sort de la boucle;

- si le pas est négatif; la variable i est décrémentée de ce pas. Si i est supérieur à vfin, l'exécution se poursuit à l'instruction suivant FOR et l'on itère encore une fois, sinon on sort de la boucle.

STEP précise la valeur du pas d'incrémentation ou de décrémentation. En l'absence de STEP, le pas prend la valeur 1 par défaut. L'utilisation de DOWNT0 en lieu de TO affecte la valeur -1 au pas et interdit l'emploi de STEP.

FUNCTION func[(par1,par2...[,VAR var1,var2...])]

@func[(par1,par2...[,VAR var1,var2...])]

par1,par2...,var1,var2... : variables.

Cette instruction déclare une fonction. Les paramètres facultatifs par1,par2... sont des variables de types quelconques (sauf de type tableau). Ces variables sont locales à la fonction. L'instruction VAR peut précéder une liste de variables (tableau y compris), celles-ci sont alors transmises par adresse et toute modification dans le corps de la fonction entraînera une modification de la variable. Une fonction se finit par l'instruction ENDFUNC, la valeur de la fonction est renvoyée par l'instruction RETURN. Il peut y avoir plusieurs RETURN dans une même fonction. L'appel se fait de la même manière qu'une fonction classique à ceci près que le signe @ (arobase) ou FN doit précéder le nom de la fonction. Les fonctions qui renvoient un résultat alphanumérique doivent porter le suffixe "\$". Les paramètres correspondant à des variables précédées par VAR dans l'en-tête de déclaration doivent être des variables ou tableaux et non pas des expressions. Une fonction peut s'appeler elle-même, on la nomme alors récursive (vous trouverez dans ce livre de nombreux exemples).

Voir également les instructions **PROCEDURE**, **LOCAL** et **VAR**.

GOSUB procédure

Appel à la procédure considérée.

GOTO label

Cette instruction oblige l'interpréteur à passer à la ligne pointée par le label. On ne l'utilise presque jamais en GFA Basic 3.0, ce serait un crime !

IF condition[/THEN/][ELSE/]/ENDIF

condition : expression numérique booléenne.

Cette instruction conditionne l'exécution de séquences d'instructions. Si la condition est vérifiée, les instructions comprises entre IF et ELSE (entre IF et ENDIF si ELSE est manquant) s'exécutent, sinon la partie comprise entre ELSE et ENDIF est exécutée (en l'absence du ELSE, on passe à la suite). Quelle que soit l'issue du test, l'exécution se poursuit à l'instruction suivant ENDIF.

Par exemple :

```
IF Rep$="oui" THEN
  PRINT "merci"
ELSE
  PRINT "dommage"
ENDIF
```

LOCAL var1[,var2...]

var1,var2... : variables.

Cette instruction déclare locales, à la procédure ou à la fonction, les variables indiquées (impossible avec un tableau). Leur portée ne dépasse pas celle de la procédure ou fonction dans laquelle elles auront été déclarées ou celle d'un niveau inférieur.

ON exp GOSUB proc1,proc2,...,procn

exp : expression numérique entière.

Cette instruction exécute la procédure correspondant au rang indiqué par la variable var : si var est compris entre 1 et 2 (2 exclu) le branchement est effectué à la procédure proc1,... Si var est compris entre n et n+1 (n+1 exclu) le branchement est effectué à la procédure procn. Si var est en dehors de l'intervalle 1..n+1 (n+1 exclu), alors aucun appel n'est effectué.

PROCEDURE proc[(par1,par2...[,VAR var1,var2...])]

proc[(par1,par2...[,VAR var1,var2...])]

par1,par2...,var1,var2... : variables.

Cette instruction déclare une procédure. Les paramètres facultatifs par1,par2.. sont des variables de types quelconques (sauf de type tableau). Ces variables sont locales à la procédure. L'instruction VAR peut précéder une liste de variables (tableau y compris), celles-ci sont alors transmises par adresse et toute modification dans le corps de la procédure entraînera une modification de la variable. Une procédure se finit par l'instruction RETURN, il ne peut y avoir plusieurs RETURN dans une même procédure.

L'appel se fait de la même manière qu'une commande classique, à ceci près que les paramètres doivent se placer entre parenthèses, ceux correspondant à une variable précédés par VAR dans l'en-tête de déclaration doivent être des variables ou tableaux et non pas des expressions. Une procédure peut s'appeler elle-même, on la nomme alors récursive (vous trouverez dans ce livre de nombreux exemples).

Voir également les instructions **FUNCTION**, **LOCAL** et **VAR**.

REPEAT/UNTIL condition

condition : expression numérique booléenne.

Exécute, jusqu'à ce que la condition soit vérifiée, toutes les instructions comprises entre REPEAT et UNTIL. La condition est testée en fin de boucle.

RETURN

Cette instruction provoque le retour d'une procédure. Les variables locales sont perdues.

RETURN exp

exp : expression.

Cette fonction renvoie l'expression exp à la fonction appelante. Les variables locales sont perdues.

SELECT var/ENDSELECT

var : variable.

Cette instruction appelée instruction de test universelle permet de conditionner l'exécution de telle ou telle suite d'instructions en fonction de la valeur de la variable entière ou alphanumérique. Cette dernière est convertie en une valeur numérique d'après les codes ASCII des caractères (le calcul du manuel est faux) en commençant par la dernière lettre. Prenons un exemple :

"ADCB" est codé avec la valeur suivante :

$66+2^8*67+2^{16}*68+2^{24}*65$

B C D A

"A" est codé avec la valeur 65.

On peut ainsi mélanger des tests sur les constantes alphanumériques et numériques.

Les divers cas sont déterminés par l'instruction CASE. Elle peut prendre plusieurs formes :

CASE constante

teste si la variable de sélection est égale à la constante ;

CASE constante1 TO constante2

teste si la variable de sélection est dans l'intervalle considéré ;

CASE constante1, constante2, constante3

teste si la variable de sélection est dans la liste des constantes ;

ou une combinaison des trois ;

CASE constante1, constante2 TO constante3, constante4

teste chacun des cas précédents.

Lorsque le test indiqué par CASE réussit, la séquence d'instructions située entre ce même CASE et le suivant, ou DEFAULT ou bien encore ENDSELECT, est exécutée. Le programme se poursuit alors après l'instruction ENDSELECT. Si la clause CONT est spécifiée, l'instruction CASE ou DEFAULT suivante sera ignorée.

Si l'instruction DEFAULT est présente, la séquence d'instructions située entre DEFAULT et ENDSELECT sera exécutée lorsque aucun test n'a réussi.

VAR

Ce mot clef présent dans l'en-tête d'une procédure ou d'une fonction devance les variables externes qui seront modifiées dans le corps de la procédure ou de la fonction. L'instruction VAR est la seule manière de pouvoir passer un tableau en paramètre. Voir également les instructions FUNCTION et PROCEDURE.

WHILE condition/WEND

condition : expression numérique booléenne.

Exécute, tant que la condition est vérifiée, toutes les instructions comprises entre WHILE et WEND. La condition est testée en début de boucle.

GESTION DES ERREURS

Voici la liste des instructions traitées dans ce chapitre :

DUMP	ON BREAK GOSUB
ERR	ON ERROR
ERR\$	ON ERROR GOSUB
ERROR	RESUME
FATAL	TRACE\$
ON BREAK	TROFF
ON BREAK CONT	TRON

DUMP [code[,fichier]]

code,fichier : expressions alphanumériques.

Cette instruction permet l'affichage des variables et des labels au cours de l'exécution. L'écran peut être remplacé par un fichier si celui-ci est spécifié. L'expression code peut prendre les valeurs suivantes :

"lettre" : affichage de toutes les variables qui commencent par la lettre indiquée ;

":" : affichage de tous les labels ;

"@" : affichage de toutes les procédures et fonctions.

ERR

Cette fonction renvoie le code de l'erreur rencontrée.

ERR\$(n)

n : expression numérique entière.

Cette fonction renvoie le message, en clair, de l'erreur de code n.

ERROR n

n : expression numérique entière.

Cette instruction simule l'erreur de code n.

FATAL

Cette fonction renvoie le type de l'erreur ; 0 pour une erreur normale ou -1 pour les erreurs graves avec un environnement faussé. En général, les erreurs fatales conduisent à la destruction du programme.

ON BREAK

Cette instruction rétablit l'arrêt normal lors de l'appui simultané des touches <CONTROL>, <SHIFT> et <ALTERNATE>.

ON BREAK CONT

Cette instruction poursuit l'exécution du programme même si les touches <CONTROL>, <SHIFT> et <ALTERNATE> sont pressées.

ON BREAK GOSUB label

Cette instruction provoque le branchement du programme au label spécifié lors de l'appui simultané des touches <CONTROL>, <SHIFT> et <ALTERNATE>.

ON ERROR

Cette instruction rétablit le mode normal d'affichage des messages d'erreur.

ON ERROR GOSUB traitement

Cette instruction provoque le branchement à la procédure traitement lors de l'apparition d'une erreur. On doit répéter l'instruction à chaque traitement pour que le même processus se poursuive à la prochaine erreur.

RESUME

Cette instruction recommence l'exécution, une nouvelle fois, à partir de l'instruction qui a provoqué l'erreur.

RESUME NEXT

Cette instruction recommence l'exécution à partir de l'instruction qui suit celle ayant provoqué l'erreur.

RESUME label

Cette instruction recommence l'exécution à partir du label spécifié. Cette instruction, RESUME label, est la seule utilisable pour des erreurs fatales.

TRACE\$

Cette fonction renvoie la prochaine instruction à exécuter.

TROFF

Cette instruction désactive le traitement des erreurs.

TRON[#n]

n : expression numérique entière.

Cette instruction effectue la sortie sur le périphérique concerné (ouvert sous le numéro de canal n ou sur l'écran) de chaque instruction exécutée.

TRON traitement

Cette instruction provoque le branchement à la procédure traitement lors de l'exécution de chaque instruction.

GESTION DE LA MEMOIRE

Ce chapitre traite des instructions liées à la gestion de la mémoire. Reportez-vous aux chapitres "Variables, tableaux et pointeurs", "Interfaçage avec les autres langages" et "Fichiers" pour trouver les instructions d'accès à la mémoire.

Voici les instructions traitées dans ce chapitre :

BASEPAGE	MALLOC
BMOVE	MFREE
FRE	MSHRINK
HIMEM	RESERVE
INLINE	

BASEPAGE

Cette variable contient l'adresse de la basepage de l'interpréteur. Cette zone, longue de 256 octets, contient les informations suivantes sur l'interpréteur.

déplacement information

0	adresse du début de la TPA ;
4	adresse du premier octet la fin de la TPA+1 ;
8	adresse du texte du programme ;
12	longueur du texte ;
16	adresse de la zone des données ;
20	longueur de la zone des données ;
24	adresse de la zone BSS ;
28	longueur de la zone BSS ;
32	adresse de la zone DTA ;
36	adresse de la Basepage du programme appelant ;
40 à 43	réservé ;
44	adresse de la chaîne d'environnement ;
48 à 127	réservé ;
128	ligne d'instruction dont la longueur est située dans le premier octet.

BMOVE adr1,adr2,n

adr1,adr2 : mot long ;

n : expression numérique entière.

Cette instruction transfère n octets de l'adresse adr1 à l'adresse adr2. Lorsque les zones mémoires se chevauchent, la copie s'effectue sans écrasement.

FRE(x)

x : expression numérique.

Cette fonction renvoie la place encore disponible en mémoire principale. Le paramètre x n'a aucune importance.

HIMEM

Cette variable contient l'adresse du premier octet libre au-dessus du GFA. Le GFA est situé entre BASEPAGE et HIMEM.

INLINE adr,n

adr : variable mot long ;

n : expression numérique entière.

Cette instruction définit une zone mémoire interne au GFA Basic qui permet d'inclure dans le corps d'un programme des routines assembleur, des images, des données ... Elle ne peut être suivie d'aucun commentaire. En effet, la zone est placée là où habituellement se placent ces derniers. L'expression n contient le nombre d'octets à réserver. La variable adr est initialisée avec l'adresse de cette zone.

Lorsque le curseur se situe sur la ligne de l'instruction **INLINE**, une ligne de menu s'affiche sur la première ligne de l'écran présentant les options suivantes :
-LOAD pour initialiser le contenu de la zone préalablement définie (l'extension prédéfinie est .INL);

-SAVE pour sauver le contenu de la zone préalablement définie (l'extension prédéfinie est .INL) ;

-CLEAR pour supprimer la zone ;

-DUMP pour effectuer l'impression de la zone en code hexadécimal ;

-ASM pour permettre de commuter avec l'assembleur lorsque celui-ci a lancé le GFA Basic (avec l'option "exécuter un programme"). Les programmes assemblés seront alors automatiquement transférés vers la zone réservée par **INLINE**.

MALLOC(n)

n : expression numérique entière.

Cette fonction (GEMDOS 72) alloue une zone libre contenant n octets et renvoie l'adresse de celle-ci. Si n vaut -1, la taille de la plus grande zone d'un seul tenant, est renvoyée. Avant de demander l'allocation de zone de taille importante, la zone du GFA Basic doit être restreinte à l'aide de l'instruction **RESERVE**. Lorsque la tentative d'allocation échoue, la fonction renvoie une valeur négative.

MFREE(adr)

adr : mot long.

Cette fonction (GEMDOS 73) libère la zone débutant à l'adresse adr et précédemment allouée avec **MALLOC**. L'adresse adr doit obligatoirement correspondre à un bloc alloué avec **MALLOC**. La valeur renvoyée est soit 0 si tout s'est bien passé, soit une valeur négative si une erreur est survenue.

MSHRINK(adr,n)

adr : mot long ;

n : expression numérique entière.

Cette fonction (GEMDOS 74) diminue la taille d'une zone mémoire débutant à l'adresse adr et précédemment allouée avec **MALLOC** en la fixant à n octets. L'adresse adr doit obligatoirement correspondre à un bloc alloué avec **MALLOC**, sinon la fonction renvoie la valeur -40. L'expression n doit être inférieure à la taille précédente, sinon la fonction renvoie la valeur -67. Si tout s'est bien déroulé, la valeur nulle est retournée.

RESERVE [n]

n : expression numérique entière.

Cette instruction fixe la taille de la zone mémoire utilisée par le GFA Basic à n octets (l'expression n doit être multiple de 256). L'absence de paramètre rétablit la situation initiale lors du lancement de l'interpréteur. Il est conseillé de contrôler n avec soin; en effet, le GEM nécessite un minimum de 16384 octets pour fonctionner correctement; l'accroissement de cette zone empiète sur celle attribuée au GEM.

HORLOGE ET INTERRUPTIONS

Voici la liste des instructions traitées dans ce chapitre :

AFTER	EVERY CONT
AFTER CONT	EVERY STOP
AFTER STOP	SETTIME
DATE\$	TIME\$
EVERY	TIMER

AFTER t GOSUB procedure

t : expression numérique entière.

Cette instruction entraîne un appel unique de la procédure lorsque le temps t est écoulé (t en 1/200 de seconde).

AFTER CONT

Cette instruction réactive l'appel de la procédure précédemment activée avec **AFTER GOSUB**, puis désactivée avec **AFTER STOP**.

AFTER STOP

Cette instruction désactive l'appel de la procédure précédemment indiquée avec **AFTER GOSUB**.

DATES

Cette fonction renvoie dans une chaîne de caractères la date au format "jj.mm.aaaa". Elle est aussi considérée comme une variable alphanumérique et peut être initialisée avec une chaîne au même format par simple affectation : **DATE\$="jj.mm.aaaa"**.

Voir également l'instruction **MODE**.

EVERY t GOSUB procedure

t : expression numérique entière.

Cette instruction entraîne une série d'appels à la procédure spécifiée. l'expression t (en 1/200 de seconde) indique la période qui s'écoule entre deux appels successifs.

EVERY CONT

Cette instruction réactive l'appel de la procédure précédemment activée avec **EVERY GOSUB**, puis désactivée avec **EVERY STOP**.

EVERY STOP

Cette instruction désactive l'appel de la procédure précédemment indiquée avec **EVERY GOSUB**.

SETTIME time,date

time,date : expressions alphanumériques.

Permet de mettre à jour l'heure et la date du système. Les variables time et date représentent respectivement des chaînes de caractères contenant l'heure (hh:mm:ss) et la date (jj.mm.aaaa). Voir également l'instruction **MODE**.

TIMES

Cette fonction renvoie l'heure courante conservée par le système dans le format "hh:mm:ss". Elle est aussi considérée comme une variable alphanumérique et peut être initialisée avec une chaîne au même format par simple affectation : `TIMES="hh:mm:ss`. Voir également l'instruction `MODE`.

TIMER

Cette fonction détermine le temps écoulé depuis la mise en route du système. La valeur donnée est exprimée en 1/200 de seconde.

GESTION DU JOYSTICK

Seules une instruction et deux fonctions permettent de manipuler la manette de jeu.

STICK mode

mode : expression numérique entière.

Cette instruction permet de choisir la nature du périphérique connecté au port 0 soit mode est égal à 0 : dans ce cas on teste la souris, soit mode est égal à 1 et alors on teste le joystick.

STICK(port)

port : expression numérique entière.

Cette fonction renvoie la position du joystick connecté au port considéré (0 ou 1) de la manière suivante :

```
5   1   9
 \   |   /
4--0--8
 /   |   \
6   2  10
```

STRIG(port)

Cette fonction renvoie l'état du bouton de feu du joystick connecté au port considéré (0 ou 1) : TRUE, si le bouton est actionné, ou FALSE dans le cas contraire.

GENERATION DE SONS

Ce chapitre se compose uniquement de deux instructions qui pilotent le processeur sonore de l'Atari ST : SOUND et WAVE.

SOUND canal,volume,note,octave[,durée]

canal,volume,note : expressions numériques entières ;

octave,durée : expressions numériques entières.

Cette instruction permet d'émettre un son en fonction des paramètres donnés.

L'Atari ST possède trois canaux numérotés de 1 à 3. Le volume est compris entre 0 et 15. Les notes s'échelonnent entre 1 et 12 :

- 1 = do ;
- 2 = do# ;
- 3 = ré ;
- 4 = ré# ;
- 5 = mi ;
- 6 = fa ;
- 7 = fa# ;
- 8 = sol ;
- 9 = sol# ;
- 10 = la ;
- 11 = la# ;
- 12 = si.

Huit octaves sont disponibles (1 à 8). La durée est facultative, elle est exprimée en 1/50 de seconde. Elle indique le temps d'attente avant l'exécution de la prochaine instruction. En général, on place un SOUND 1,0 pour stopper un son sur le canal 1.

Par exemple, SOUND 3,11,10,4,1000 reproduit la tonalité du téléphone (LA international) pendant vingt secondes.

L'instruction SOUND peut aussi se programmer en fonction de la période d'une note : SOUND canal,volume,#période[,durée]. On calcule la période à partir de la fréquence d'une note avec la formule suivante :

Période=Trunc(125000/Fréquence+.5).

Par exemple, SOUND 2,12,#284,1000 produit exactement la même note pendant la même durée que l'exemple précédent.

WAVE canal,enveloppe,forme,période,durée

canal,enveloppe,forme : expressions numériques entières ;

période,durée : expressions numériques entières.

Cette instruction permet de faire des effets sonores en combinant les trois canaux du ST. Le canal est codé en binaire, chaque bit ayant une attribution :

- 1 = canal 1 activé (bit 0) ;
- 2 = canal 2 activé (bit 1) ;
- 4 = canal 3 activé (bit 2) ;
- 8 = bruit sur le canal 1 (bit 3) ;
- 16 = bruit sur le canal 2 (bit 4) ;
- 32 = bruit sur le canal 3 (bit 5).

Le paramètre "enveloppe" spécifie de manière identique au paramètre "canal", les canaux modulés par le générateur d'enveloppe. La forme de l'enveloppe

codée entre 0 et 15 correspond au tableau suivant :

- 0, 1, 2, 3, 9 = linéaire décroissant ;
- 4, 5, 6, 7, 15 = linéaire décroissant, retombant brusquement ;
- 11 = linéaire décroissant, puis saut brusque à une valeur haute ;
- 13 = linéaire croissant constant ;
- 8 = dents de scie décroissantes ;
- 12 = dents de scie croissantes ;
- 10 = triangle décroissant au début ;
- 14 = triangle croissant au début.

BITS du REGISTRE R15				REPRESENTATION GRAPHIQUE DES SORTIES DU GENERATEUR D'ENVELOPPE E2 E1 E0
CONT- NUITE	ATTNOM	ALTER- SOUNDCE	RETEMAR	
0	0	X	X	
0	1	X	X	
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	

La période est celle de l'enveloppe. La période correspond musicalement à des vibratos et des trémolos. La durée, exprimée en cinquantième de seconde, indique le temps d'attente avant l'exécution de la prochaine instruction.

INTERFACAGE AVEC LES AUTRES LANGAGES

Voici la liste des instructions traitées dans ce chapitre :

C;
CALL
EXEC
MONITOR
RCALL

C:adr([par1,par2...])

adr : mot long ;

par1,par2... mots ou mots longs.

Cette fonction appelle la routine en lui passant par la pile une liste de paramètres qui sont soit des mots précédés par W: (pris par défaut), soit des mots longs précédés par L:. Toute erreur dans le passage des paramètres, portant sur leur valeur ou leur type (L: ou W:), peut entraîner un crash général de la machine. Au retour, elle renvoie la valeur de retour ou le code d'erreur, précédemment stocké dans D0.

CALL adr([par1,par2...])

adr : mot long ;

par1,par2... mots ou mots longs.

Cette fonction appelle la routine en lui passant par la pile le nombre de paramètres et l'adresse de la liste de paramètres qui sont soit des mots précédés par W: (pris par défaut), soit des mots longs précédés par L:. Toute erreur dans le passage des paramètres, portant sur leur valeur ou leur type (L: ou W:), peut entraîner un crash général de la machine. Au retour, elle renvoie la valeur de retour ou le code d'erreur, précédemment stocké dans D0.

EXEC mode,fichier,commande,environnement

mod : expression numérique entière ;

fichier,commande,environnement : expressions alphanumériques.

Cette instruction ou fonction charge et lance des programmes qui rendront le contrôle au programme appelant. Le paramètre mode peut prendre deux valeurs : 0 qui signifie charger et exécuter et 3 qui signifie charger seulement.

MONITOR [n]

Cette instruction provoque une instruction illégale du MC68000 (code \$4AFC) et un branchement à la routine pointée par le vecteur Illégal Instruction (vecteur \$04) si ce dernier a été modifié par un moniteur. Le paramètre facultatif n est chargé dans le registre D0.

RCALL adr,tab

adr : mot long ;

tab : tableau de mots longs.

Cette fonction appelle la routine d'adresse adr en initialisant les registres D0 à D7 et A0 à A6 avec respectivement les valeurs éléments du tableau de mots longs d'indice 0 à 15. Au retour, elle renvoie la valeur de retour ou le code d'erreur, précédemment stocké dans D0.

**EXEMPLE DE
COMMUNICATION :
GFA BASIC
ET ASSEMBLEUR**

Chaque programme est donné dans son intégralité et décortiqué afin de vous expliquer comment fonctionne les diverses instructions. Les programmes fonctionnent dans la mesure du possible dans les trois résolutions de l'Atari ST.

Addition et multiplication de matrices

Ce programme effectue des additions et des multiplications de matrices

Programme principal

Les instructions suivantes sont facultatives; toutefois, elles faciliteront l'écriture du programme. En effet, lorsque les variables sont saisies (mode DEFLIST 2), elles prennent automatiquement le bon suffixe à l'affichage.

```
DEFBYT "i,j,k,nb,c,l"
```

```
DEFWRD "mat"
```

Variables globales :

adr_addq %	adresse de la routine d'addition ;
adr_mulq %	adresse de la routine de multiplication ;
matrice1&()	première matrice ;
matrice2&()	seconde matrice ;
matrice3&()	matrice contenant le résultat.

Procédure mat_input

Cette procédure initialise les matrices.

Paramètres :

nbl 	nombre de lignes ;
nbc 	nombre de colonnes ;
mat&()	matrice.

Variables locales :

i j 	indices de boucle.
-------------	--------------------

Procédure mat_print

Cette procédure affiche la matrice spécifiée.

Paramètres :

l 	ligne de début d'affichage ;
c 	colonne de début d'affichage ;
mat&()	matrice à afficher.

Variables locales :

i j 	indices de boucle ;
nbl 	nombre de lignes ;
nbc 	nombre de colonnes ;
l_a 	ligne d'affichage ;
c_a 	colonne d'affichage.

Fonction mat_mul

Cette fonction multiplie deux matrices.

Paramètres :

mat1&()	première matrice ;
mat2&()	seconde matrice ;
mat3&()	matrice contenant le résultat.

Variables locales :

i j k 	indices de boucle ;
nbl1 	nombre de lignes de la première matrice ;
nbc1 	nombre de colonnes de la première matrice ;
nbl2 	nombre de lignes de la seconde matrice ;
nbc2 	nombre de colonnes de la seconde matrice ;
flag 	indique si l'opération est possible.

Fonction mat_mulq

Cette fonction multiplie deux matrices en langage assembleur.

Paramètres :

mat1&()	première matrice ;
mat2&()	seconde matrice ;
mat3&()	matrice contenant le résultat.

Variables locales :

i j k 	indices de boucle ;
nbl1 	nombre de lignes de la première matrice ;
nbc1 	nombre de colonnes de la première matrice ;
nbl2 	nombre de lignes de la seconde matrice ;
nbc2 	nombre de colonnes de la seconde matrice ;
flag 	indique si l'opération est possible.

Fonction mat_add

Cette fonction additionne deux matrices.

Paramètres :

mat1&()	première matrice ;
mat2&()	seconde matrice ;
mat3&()	matrice contenant le résultat.

Variables locales :

i j k 	indices de boucle ;
nbl1 	nombre de lignes de la première matrice ;
nbc1 	nombre de colonnes de la première matrice ;
nbl2 	nombre de lignes de la seconde matrice ;
nbc2 	nombre de colonnes de la seconde matrice ;
flag 	indique si l'opération est possible.

Fonction mat_addq

Cette fonction additionne deux matrices en langage assembleur.

Paramètres :

mat1&()	première matrice ;
mat2&()	seconde matrice ;
mat3&()	matrice contenant le résultat.

Variables locales :

i j k 	indices de boucle ;
nbl1 	nombre de lignes de la première matrice ;
nbc1 	nombre de colonnes de la première matrice ;
nbl2 	nombre de lignes de la seconde matrice ;
nbc2 	nombre de colonnes de la seconde matrice ;
flag 	indique si l'opération est possible.

Procédure affiche_matrice

Cette procédure affiche les matrices et le résultat de l'opération.

Paramètres :

mat1&() première matrice ;
mat2&() seconde matrice ;
mat3&() matrice contenant le résultat.

Variables locales :

nbl1| nombre de lignes de la première matrice ;
nbc1| nombre de colonnes de la première matrice ;
nbl2| nombre de lignes de la seconde matrice ;
nbc2| nombre de colonnes de la seconde matrice.

Programme :

```

-----
|          MATRICES          |
|-----|
|
|  DEFBYT "i,j,k,nb,c,l"
|  DEFWRD "mat"
|
|  INLINE adr_addq%,520
|  INLINE adr_mulq%,520
|  mat_input(4,3,matrice1&())
|  mat_input(3,3,matrice2&())
|  IF @mat_mul(matrice1&(),matrice2&(),matrice3&()) THEN
|      affiche_matrices(matrice1&(),matrice2&(),matrice3&())
|  ENDIF
|  ~INP(2)
|  RESTORE
|  mat_input(4,3,matrice1&())
|  mat_input(3,3,matrice2&())
|  IF @mat_mulq(matrice1&(),matrice2&(),matrice3&()) THEN
|      affiche_matrices(matrice1&(),matrice2&(),matrice3&())
|  ENDIF
|  ~INP(2)
|  RESTORE
|  mat_input(4,3,matrice1&())
|  mat_input(4,3,matrice2&())
|  IF @mat_add(matrice1&(),matrice2&(),matrice3&()) THEN
|      affiche_matrices(matrice1&(),matrice2&(),matrice3&())
|  ENDIF
|  ~INP(2)
|  RESTORE

```

```

mat_input(3,4,matrice1&())
mat_input(3,4,matrice2&())
IF @mat_addq(matrice1&(),matrice2&(),matrice3&()) THEN
    affiche_matrices(matrice1&(),matrice2&(),matrice3&())
ENDIF
~INP(2)
END

' -----
'      PROCEDURES ET FONCTIONS
' -----

PROCEDURE mat_input(nbl|,nbc|,VAR mat&())
    LOCAL i|,j|
    ERASE mat&()
    DIM mat&(nbl|,nbc|)
    FOR i|=1 TO nbl|
        FOR j|=1 TO nbc|
            READ mat&(i|,j|)
        NEXT j|
    NEXT i|
RETURN
'

PROCEDURE mat_print(c|,l|,VAR mat&())
    LOCAL i|,j|,c_a|,l_a|,nbl|,nbc|
    nbl|=SUB(BYTE({ADD({*mat&() },4)}),1)
    nbc|=SUB(BYTE({*mat&() })),1)
    FOR i|=1 TO nbl|
        PRINT AT(ADD(c|,3),ADD(l|,MUL(i|,2))); " ("
        PRINT AT(ADD(c|,MUL(nbc|,6)),ADD(l|,MUL(i|,2))); " "
        IF i|<nbl| THEN
            PRINT AT(ADD(c|,3),ADD(ADD(l|,1),MUL(i|,2))); " ("
            PRINT
        AT(ADD(c|,MUL(nbc|,6)),ADD(ADD(l|,1),MUL(i|,2))); " "
        ENDIF
        FOR j|=1 TO nbc|
            c_a|=ADD(c|,MUL(j|,5))
            l_a|=ADD(l|,MUL(i|,2))
            PRINT AT(c_a|,l_a|); USING "###",mat&(i|,j|)
        NEXT j|
    NEXT i|
RETURN
'

FUNCTION mat_mul(VAR mat1&(),mat2&(),mat3&())
    LOCAL i|,j|,k|,nbl1|,nbc1|,nbl2|,nbc2|,flag|
    nbl1|=SUB(BYTE({ADD({*mat1&() },4)}),1)
    nbc1|=SUB(BYTE({*mat1&() })),1)
    nbl2|=SUB(BYTE({ADD({*mat2&() },4)}),1)
    nbc2|=SUB(BYTE({*mat2&() })),1)

```

```

IF nbc1=nb12 THEN
  ERASE mat3&()
  DIM mat3&(nb11|,nbc2|)
  FOR i|=1 TO nb11|
    FOR j|=1 TO nbc2|
      FOR k|=1 TO nbc1|
        ADD mat3&(i|,j|),MUL(mat1&(i|,k|),mat2&(k|,j|))
      NEXT k|
    NEXT j|
  NEXT i|
  flag!=TRUE
ELSE
  ALERT 1,"Produit de matrices impossible",1,"Vu",b%
  flag!=FALSE
ENDIF
RETURN flag!
ENDFUNC
'
FUNCTION mat_mulq(VAR mat1&(),mat2&(),mat3&())
  LOCAL i|,j|,nb11|,nbc1|,nb12|,nbc2|,flag!
  nb11|=SUB(BYTE({ADD({*mat1&() },4)}),1)
  nbc1|=SUB(BYTE({{*mat1&() }}),1)
  nb12|=SUB(BYTE({ADD({*mat2&() },4)}),1)
  nbc2|=SUB(BYTE({{*mat2&() }}),1)
  IF nbc1=nb12 THEN
    ERASE mat3&()
    DIM mat3&(nb11|,nbc2|)
    ~C:adr_mulq%(L:*mat1&(),L:*mat2&(),L:*mat3&())
    flag!=TRUE
  ELSE
    ALERT 1,"Multiplication de matrices impossible",1,"Vu",b%
    flag!=FALSE
  ENDIF
  RETURN flag!
ENDFUNC
'
FUNCTION mat_add(VAR mat1&(),mat2&(),mat3&())
  LOCAL i|,j|,nb11|,nbc1|,nb12|,nbc2|,flag!
  nb11|=SUB(BYTE({ADD({*mat1&() },4)}),1)
  nbc1|=SUB(BYTE({{*mat1&() }}),1)
  nb12|=SUB(BYTE({ADD({*mat2&() },4)}),1)
  nbc2|=SUB(BYTE({{*mat2&() }}),1)
  IF nbc1=nbc2 AND nb11=nb12 THEN
    ERASE mat3&()
    DIM mat3&(nb11|,nbc1|)
    FOR i|=1 TO nb11|
      FOR j|=1 TO nbc1|

```

```

        mat3&(i|,j|)=ADD(mat1&(i|,j|),mat2&(i|,j|))
    NEXT j|
NEXT i|
flag!=TRUE
ELSE
    ALERT 1,"Addition de matrices|impossible",1,"Vu",b%
    flag!=FALSE
ENDIF
RETURN flag!
ENDFUNC
'
FUNCTION mat_addq(VAR mat1&(),mat2&(),mat3&())
    LOCAL i|,j|,nbl1|,nbc1|,nbl2|,nbc2|,flag!
    nbl1|=SUB(BYTE(({ADD(({*mat1&()}),4)}),1)
    nbc1|=SUB(BYTE(({*mat1&()})),1)
    nbl2|=SUB(BYTE(({ADD(({*mat2&()}),4)}),1)
    nbc2|=SUB(BYTE(({*mat2&()})),1)
    IF nbc1|=nbc2| AND nbl1|=nbl2| THEN
        ERASE mat3&()
        DIM mat3&(nbl1|,nbc1|),reg%(15)
        reg%(1)=MUL(nbc1|,nbl1|)+(nbc1|-1)
        reg%(9)=V:mat1&(1,1)
        reg%(10)=V:mat2&(1,1)
        reg%(11)=V:mat3&(1,1)
        RCALL adr_addq%,reg%()
        flag!=TRUE
    ELSE
        ALERT 1,"Addition de matrices|impossible",1,"Vu",b%
        flag!=FALSE
    ENDIF
    RETURN flag!
ENDFUNC
'
PROCEDURE affiche_matrices(VAR mat1&(),mat2&(),mat3&())
    LOCAL nbl1|,nbc1|,nbl2|,nbc2|
    nbl1|=SUB(BYTE(({ADD(({*mat1&()}),4)}),1)
    nbc1|=SUB(BYTE(({*mat1&()})),1)
    nbl2|=SUB(BYTE(({ADD(({*mat2&()}),4)}),1)
    nbc2|=SUB(BYTE(({*mat2&()})),1)
    CLS
    mat_print(1,ADD(1,MUL(nbl2|,2)),mat1&())
    mat_print(ADD(4,MUL(nbc1|,5)),1,mat2&())
    mat_print(ADD(4,MUL(nbc1|,5)),ADD(1,MUL(nbl2|,2)),mat3&())
RETURN
'
donnee:
DATA 1,-2,5,0,6

```



```

DATA -6,8,9,1,7
DATA -1,0,0,2,2
DATA 11,0,-3,2,-1
DATA 1,8,-6,7,-2
DATA 0,0,-1,1,1

```

Voici les codes sources des routines en assembleur.

```

;                                     Addition de matrices
;
; .TEXT
LIGNE: move.w (a1)+,d0 ; élément dans d0
      add.w (a2)+,d0 ; on additionne le second
      move.w d0,(a3)+ ; somme dans le troisième tableau
      dbra d1,LIGNE ; on boucle
; fin
      rts ; retour au basic
      .END

```

```

;                                     Multiplication de matrices
;
; .TEXT
;initialisation
      movea.l 4(sp),a1 ; adresse du descripteur
      movea.l 8(sp),a2 ; " "
      movea.l 12(sp),a3 ; " "
      movea.l (a1),a1 ; adresse du tableau
      movea.l (a2),a2 ; " "
      movea.l (a3),a3 ; " "
      addq.l #4,a1 ; pour avoir le nombre de lignes
      move.l (a1)+,d0 ; nombre de lignes mat1
      move.l (a2)+,d2 ; nombre de colonnes mat2
      move.l (a2)+,d1 ; nombre de lignes mat2
      subq.l #1,d0 ; pour tester d0 à 0
      subq.l #1,d1 ; pour tester d1 à 0
      subq.l #1,d2 ; pour tester d2 à 0
      addq.l #8,a3 ; pointe sur le premier élément
      link a6,#-16 ; réserve 16 octets
      movem.l d0/d1/a1/a3,(sp) ; empile dans l'espace

```

```

COLONNE2:
      move.l a3,-4(a6) ; sauve a3
      movea.l -8(a6),a1 ; restaure a1

      move.l -12(a6),d1 ; initialise d1
LIGNE2: movea.l -4(a6),a3 ; restaure a3
      move.w (a2)+,d3 ; prend le coefficient de mat2

      move.l -16(a6),d0 ; initialise d0

```

```
LIGNE1: move.w (a1)+,d4 ; prend le coefficient de mat1
        muls.w d3,d4    ; et multiplie
        add.w d4,(a3)+   ; additionne à la valeur de mat3
        dbra d0,LIGNE1   ; boucle sur la ligne de mat1
        dbra d1,LIGNE2   ; boucle sur la ligne de mat2
        dbra d2,COLONNE2 ; boucle sur la colonne de mat2
; fin
        unlk a6          ; restaure la pile
        rts              ; retour au basic
.END
```

GESTION DU CLAVIER

Voici la liste des instructions traitées dans ce chapitre :

KEYDEF	KEYGET
KEYLOOK	KEYPAD
KEYPRESS	KEYTEST

KEYDEF n,ch\$

n : expression numérique entière ;

ch\$: expression alphanumérique.

Cette instruction définit la touche de fonction avec la chaîne de caractères spécifiée d'une longueur maximale de 31 caractères. Les touches F1 à F10 sont respectivement codées de 1 à 10 et de 11 à 20 lorsque la touche <SHIFT> est simultanément pressée. L'affectation de ces touches reste valable sous l'éditeur du GFA Basic, toutefois la touche <ALTERNATE> doit également être pressée.

KEYGET n

n : variable mot long.

Cette instruction attend que l'utilisateur presse sur une touche. Au retour, n contient le code ASCII dans les bits 0 à 7, le code clavier dans les bits 16 à 23, et l'état des touches de contrôle du clavier.

KEYLOOK n

n : variable mot long.

Cette instruction lit la mémoire clavier d'une manière identique à KEYGET, mais sans modifier le tampon en aucune manière.

KEYPAD n

n : expression numérique entière.

Cette instruction gère les touches de contrôle du clavier.

Les cinq premiers bits de n ont la signification suivante :

bit	signification	0	1
0	NUM-LOOK	désactivé	activé ;
1	NUM-LOOK	inactivable	activable ;
2	CTRL-KEYPAD	normal	curseur ;
3	ALT-KEYPAD	normal	entrée ASCII ;
4	KEYDEF sans ALT	désactivé	activé ;
5	KEYDEF avec ALT	désactivé	activé.

KEYPRESS n

n : variable mot long.

Cette instruction simule l'appui sur la touche de code n (code ASCII et code clavier).

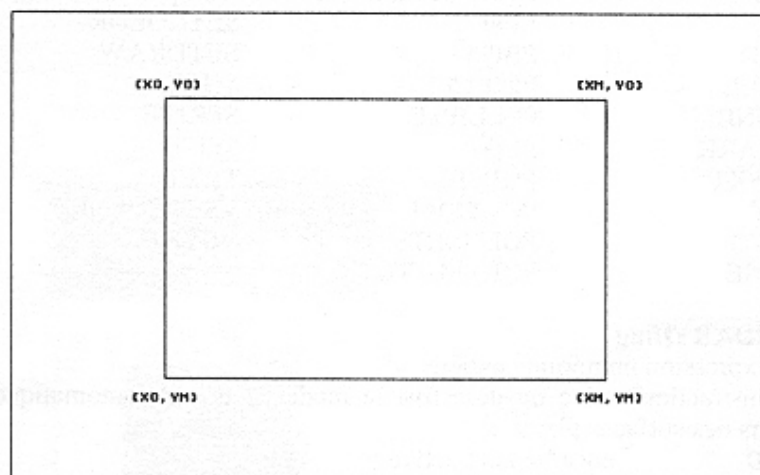
KEYTEST n

n : variable mot long.

Cette fonction renvoie le code ASCII de la touche pressée (les touches de contrôle ne sont pas prises en compte).

GRAPHISME

L'Atari ST dispose de trois modes de résolution graphique : la basse, la moyenne et la haute. Dans chacune de ces modalités, les coordonnées maximales en X ou en Y varient. De même, le nombre de couleurs de la palette varie lui aussi.



Dans toutes les résolutions, le coin supérieur gauche aura les mêmes coordonnées : (0,0). Il n'en va pas de même pour les coordonnées des autres angles.

En basse résolution (320x200) :

XO = 0 et XM = 319 ;

YO = 0 et YM = 199 ;

16 couleurs.

En moyenne résolution (640x200) :

XO = 0 et XM = 639 ;

YO = 0 et YM = 199 ;

4 couleurs.

En haute résolution (640x400) :

XO = 0 et XM = 639 ;

YO = 0 et YM = 399 ;

monochrome.

Voici les instructions traitées dans ce chapitre :

BOUNDARY	FILL	PRBOX
BOX	GET	PUT
CIRCLE	GRAPHMODE	RBOX
CLIP	LINE	SETCOLOR
COLOR	PBOX	SETDRAW
DEFFILL	PCIRCLE	SGET
DEFLINE	PELLIPSE	SPRITE
DEFMARK	PLOT	SPUT
DEFTXT	POINT()	TEXT
DRAW	POLYFILL	VSETCOLOR
DRAW()	POLYLINE	VSYNC
ELLIPSE	POLYMARK	

BOUNDARY flag

flag : expression numérique entière.

Cette instruction active ou désactive le mode de traçage automatique des contours des surfaces pleines.

flag = 0 encadrement activé ;

flag <> 0 encadrement désactivé.

BOX x1,y1,x2,y2

x1,y1,x2,y2 : expressions numériques entières.

Cette instruction dessine un rectangle entre les points de coordonnées (x1,y1) et (x2,y2) qui constituent respectivement le coin supérieur gauche et le coin inférieur droit du rectangle.

Voir également les instructions **PBOX**, **PRBOX** et **RBOX**.

CIRCLE x,y,r [alpha1,alpha2]

x,y,r,alpha1,alpha2 : expressions numériques entières.

Cette instruction dessine un cercle ou un arc de cercle de centre le point de coordonnées (x,y) et de rayon r. Les angles alpha1 et alpha2 indiquent pour un arc de cercle l'angle de début et celui de fin exprimés en 1/10 de degré.

Voir également l'instruction **PCIRCLE**.

CLIP x1,y1,lageur,hauteur [OFFSET x0,y0]

CLIP x1,y1 TO x2,y2 [OFFSET x0,y0]

x0,y0,x1,y1,x2,y2,lageur, hauteur : expressions numériques entières;

Cette instruction permet de limiter les opérations graphiques VDI à la portion rectangulaire de l'écran spécifiée par le coin supérieur gauche (x1,y1) suivi de la largeur et la hauteur de cette zone, ou par les deux angles diagonalement opposés (x1,y1) et (x2,y2). Le mot clef optionnel **OFFSET** fixe les nouvelles coordonnées de l'origine des axes X et Y.

CLIP OFFSET x0,y0

x0,y0 : expressions numériques entières.

Cette instruction permet de fixer les nouvelles coordonnées de l'origine des axes X et Y.

CLIP #n [OFFSET x0,y0]

x0,y0,n : expressions numériques entières.

Cette instruction fixe les opérations graphiques aux dimensions de la fenêtre numéro "n".

CLIP OFF

Cette instruction désactive le mode de "clipping" mis en place par l'instruction CLIP.

COLOR c

c : expression numérique entière.

Cette instruction permet de choisir la couleur d'écriture "c". Le paramètre "c" varie de une plage de 0 à 15 en basse résolution, de 0 à 3 en moyenne résolution, et de 0 à 1 en haute résolution.

Voir également les instructions VSETCOLOR et SETCOLOR.

DEFFILL [c][,a][,b]**DEFFILL [c,] var\$**

a,b,c : expressions numériques entières ;

var\$: variable alphanumérique.

Cette instruction a une double syntaxe. Elle permet dans le premier cas de définir la couleur "c" et le motif du remplissage avec les paramètres "a" et "b".

Le paramètre "a" indique le mode de remplissage :

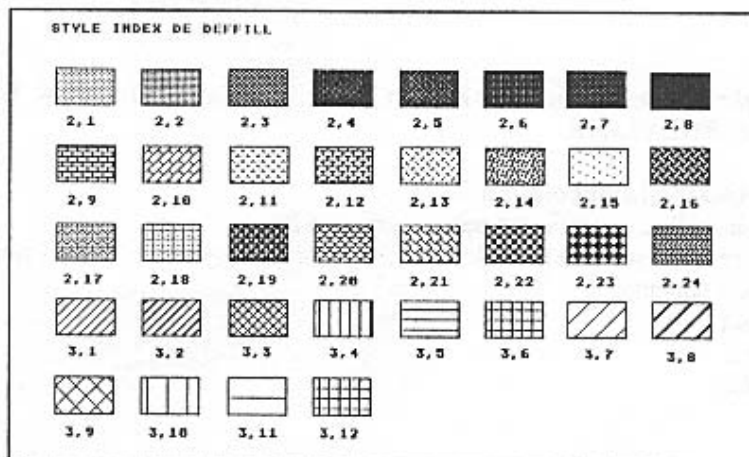
0 = vide ;

1 = plein ;

3 = hachuré ;

4 = défini par l'utilisateur.

Le paramètre "b" spécifie le motif choisi parmi vingt-quatre types dans les pointillés et parmi douze dans les hachurés.



Pour définir un motif propre, il faut constituer une matrice 16x16. La variable alphanumérique "var\$" doit résulter de seize concaténations successives de seize bits transformés en chaînes de caractères par la fonction MKIS en haute résolution (un plan), de trente-deux concaténations en moyenne résolution (deux plans) et de soixante-quatre concaténations en basse résolution (quatre plans).

Les paramètres étant facultatifs, il est indispensable de mettre les virgules séparatrices en lieu et place de ceux qui sont omis.

Voir également les instructions **FILL**, **MKIS**, **PBOX**, **PCIRCLE**, **PELLIPSE** et **POLYFILL**.

DEFLINE [type] [,largeur] [,tdébut] [,tfin]

type,largeur,tdébut,tfin : expressions numériques entières.

Cette instruction spécifie les lignes utilisées par les instructions **BOX**, **CIRCLE**, **DRAW**, **ELLIPSE**, **LINE**, **RBOX** et **POLYLINE**. Le paramètre "type" détermine le type de trait, "largeur" indique la largeur en nombre de points graphiques (pixels), "tdébut" et "tfin" donnent respectivement la forme d'extrémité à utiliser en début et fin de ligne :

0 = normal ;

1 = flèche ;

2 = arrondi.

—————	DEFFILL 1, 1
- - - - -	DEFFILL 2, 1
.	DEFFILL 3, 1
- . - . -	DEFFILL 4, 1
- - - - -	DEFFILL 5, 1
.	DEFFILL 6, 1

Voir également les instructions **BOX**, **CIRCLE**, **DRAW**, **ELLIPSE**, **LINE**, **RBOX** et **POLYLINE**.

DEFMARK [c],[symbole],[taille]

c,symbole,taille : expressions numériques entières.

Cette instruction spécifie la couleur "c", le paramètre "symbole" pourra prendre les valeurs suivantes :

1 = point ;

2 = plus ;

3 = étoile ;

4 = rectangle ;

5 = croix ;

6 = losange.

Le paramètre "taille" est la dimension en pixels du symbole utilisé lors de l'instruction POLYMARK. La taille peut prendre les valeurs 0, 20, 40, 60, 80 et 100.

Voir également l'instruction **POLYMARK**.

DEFTEXT [c],[type],[alpha],[hauteur]

c,type,alpha,hauteur : expressions numériques entières.

Cette instruction permet de définir la couleur "c", le "type" de caractères :

0 = normal ;

1 = gras ;

2 = clair ;

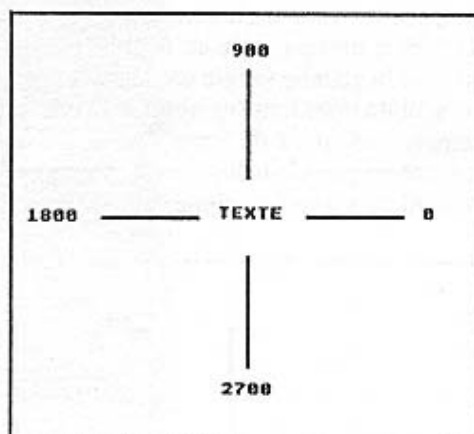
4 = italique ;

8 = souligné ;

16 = contour.

Pour avoir plusieurs caractéristiques simultanément, il suffit d'additionner les valeurs des types fondamentaux : par exemple, écriture claire et oblique ; type sera 4+2=6.

L'angle d'écriture est exprimé en 1/10 de degré.



Le paramètre "hauteur" détermine la hauteur en pixel des caractères :

4 = caractères contenus dans les icônes ;

6 = caractères fins allongés ;

13 = caractères normaux ;

32 = grands caractères.

Toutes les valeurs sont admises, mais rares sont celles qui permettent une lecture aisée.

Voir également les instructions **TEXT** et **COLOR**.

DRAW [x0,y0] [TO] [x1,y1] [TO x2,y2 ...]

x0,y0,x1,y1,x2,y2 : expressions numériques entières.

Cette instruction affiche un point ou trace des lignes entre deux ou plusieurs points. DRAW x,y est identique à PLOT. DRAW TO x,y trace une droite entre le dernier point affiché par DRAW, PLOT, LINE. DRAW x0,y0 TO x1,y1 TO ... TO xn,yn relie dans l'ordre indiqué les n points par des droites.

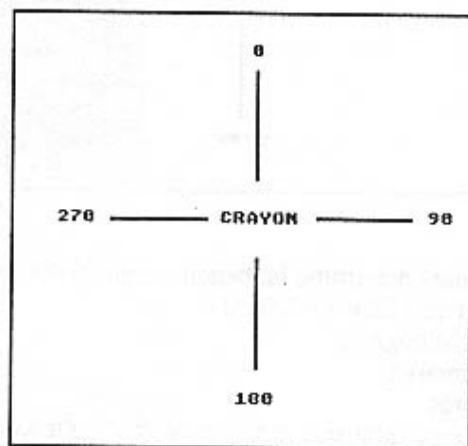
Voir également les instructions PLOT et LINE.

DRAW expression

Cette instruction simule le mouvement de la tortue du langage Logo. Il s'agit des actions d'un "crayon" sur le papier. L'expression contiendra une séquence de commandes séparées par des virgules, des points-virgules ou des apostrophes.

Liste des commandes :

FD n	avance de n pas ;
BK n	recule de n pas ;
SX x	modifie la graduation sur X utilisée par FD et BK ;
SY y	modifie la graduation sur Y utilisée par FD et BK ;
SX 0	désactive la graduation courante sur X ;
SY 0	désactive la graduation courante sur Y ;
CO coul	choisit la couleur de tracé ;
PD	baisse le crayon sur l'écran ;
PU	lève le crayon de l'écran ;
TT ang	rotation de la valeur d'angle (en degrés) spécifiée ;
RT ang	rotation vers la droite (angle en degrés) ;
LT ang	rotation vers la gauche (angle en degrés) ;
MA x,y	déplacement en coordonnées absolues vers (x,y) ;
DA x,y	déplacement avec tracé de ligne ;
MR x,y	déplacement en coordonnées relatives vers (x,y) ;
DR x,y	déplacement avec tracé de ligne.



Voir également les instructions COLOR, DRAW() et SETDRAW.

DRAW(n)

n : expression numérique entière.

Cette fonction retourne la valeur en virgule flottante des coordonnées (x,y), l'état du crayon ou des angles utilisés par l'instruction **DRAW** précédemment décrite.

Le paramètre "n" peut prendre les valeurs suivantes :

- n = 0 abscisse x ;
- n = 1 ordonnée y ;
- n = 2 angle en degrés ;
- n = 3 graduation sur X ;
- n = 4 graduation sur Y ;
- n = 5 état du crayon (0 = levé et -1 = baissé).

Voir également les instructions **DRAW** et **SETDRAW**.

ELLIPSE x,y,rx,ry [,alpha1,alpha2]

x,y,rx,ry,alpha1,alpha2 : expressions numériques entières.

Cette instruction dessine une ellipse ou un arc d'ellipse de centre le point de coordonnées (x,y) de demi-axe horizontal rx et de demi-axe vertical ry. Les angles alpha1 et alpha2 indiquent pour un arc d'ellipse l'angle de début et celui de fin exprimés en 1/10 de degré.

Voir également l'instruction **PELLIPSE**.

FILL x,y

x,y : expressions numériques entières.

Cette instruction permet de remplir la surface convexe fermée entourant le point de coordonnées (x,y) avec le motif défini grâce à l'instruction **DEFFILL**.

Voir également l'instruction **DEFFILL**.

GET x0,y0,x1,y1,var\$

x0,y0,x1,y1 : expressions numériques entières ;

var\$: variable alphanumérique.

Cette instruction affecte à la variable "var\$" une portion rectangulaire de l'écran. Les coordonnées (x1,y1) et (x2,y2) constituent respectivement le coin supérieur gauche et le coin inférieur droit du rectangle.

Voir également les instructions **PUT**, **SGET** et **SPUT**.

GRAPHMODE n

n : expression numérique entière.

Cette instruction active le mode graphique numéro "n". Il existe quatre modes graphiques :

- n = 1 mode normal, la nouvelle image écrase l'ancienne ;
- n = 2 la nouvelle image est transparente ;
- n = 3 tout point de la nouvelle image est allumé s'il était éteint, et éteint s'il était allumé ;
- n = 4 la nouvelle image est affichée en inversion vidéo.

LINE x0,y0,x1,y1

x0,y0,x1,y1 : expressions numériques entières.

Cette instruction trace une ligne entre le point de coordonnées (x0,y0) et le point de coordonnées (x1,y1).

Voir également les instructions **DEFLINE** et **DRAW**.

PBOX x1,y1,x2,y2

x1,y1,x2,y2 : expressions numériques entières.

Instruction identique à **BOX** mais avec un rectangle plein. La couleur et le motif de remplissage sont déterminés par l'instruction **DEFFILL**.

Voir également les instructions **BOX**, **DEFFILL**, **PRBOX** et **RBOX**.

PCIRCLE x,y,r [,alpha1,alpha2]

x,y,r,alpha1,alpha2 : expressions numériques entières.

Instruction identique à **CIRCLE** mais avec un cercle ou un arc de cercle plein. La couleur et le motif de remplissage sont déterminés par l'instruction **DEFFILL**.

Voir également les instructions **CIRCLE** et **DEFFILL**.

PELLIPSE x,y,rx,ry [,alpha1,alpha2]

x,y,rx,ry,alpha1,alpha2 : expressions numériques entières.

Instruction identique à **ELLIPSE** mais avec une ellipse pleine ou un arc d'ellipse plein. La couleur et le motif de remplissage sont déterminés par l'instruction **DEFFILL**.

Voir également les instructions **DEFFILL** et **ELLIPSE**.

PLOT x,y

x,y : expressions numériques entières.

Cette instruction affiche le point de coordonnées (x,y) à l'écran.

Voir également l'instruction **DRAW**.

POINT(x,y)

x,y : expressions numériques entières.

Cette fonction teste le point de coordonnées (x,y) à l'écran et renvoie la couleur : 0 ou 1 selon que le point est éteint ou allumé en haute résolution, 0 à 3 en moyenne résolution, 0 à 15 en basse résolution.

Voir également l'instruction **COLOR**.

POLYFILL n,x(),y() [OFFSET dx,dy]

n,dx,dy : expressions numériques entières ;

x(),y() : tableau numérique entier.

Cette instruction relie n points dont les coordonnées sont contenues dans les tableaux x() et y() et remplit l'intérieur (voir **FILL**). Les coordonnées du premier point sont contenues dans x(0) et y(0).

Le mot clef facultatif **OFFSET dx,dy** permet d'effectuer une translation de vecteur (dx,dy). Cependant, cette facilité ne fonctionnait pas sur toutes les versions précédentes, en particulier sur la 2.02.

Voir également les instructions **DEFFILL** et **FILL**.

POLYLINE n,x(),y() [OFFSET dx,dy]

n,dx,dy : expressions numériques entières;

x(),y() : tableau numérique entier.

Cette instruction relie n points dont les coordonnées sont contenues dans les tableaux x() et y(). Les coordonnées du premier point sont contenues dans x(0) et y(0).

Le mot clef facultatif **OFFSET dx,dy** permet d'effectuer une translation de vecteur (dx,dy). Cependant, cette facilité ne fonctionnait pas sur toutes les versions précédentes, en particulier sur la version 2.02.

Voir également les instructions **DEFLINE** et **LINE**.

POLYMARK n,x(),y() [OFFSET dx,dy]

n,dx,dy : expressions numériques entières;

x(),y() : tableau numérique entier.

Cette instruction marque, d'un symbole défini grâce à **DEFMARK**, n points dont les coordonnées sont contenues dans les tableaux x() et y(). Les coordonnées du premier point sont contenues dans x(0) et y(0).

Le mot clef facultatif **OFFSET dx,dy** permet d'effectuer une translation de vecteur (dx,dy). Cependant, cette facilité ne fonctionnait pas sur toutes les versions précédentes, en particulier sur la 2.02.

Voir également l'instruction **DEFMARK**.

PRBOX x1,y1,x2,y2

x1,y1,x2,y2 : expressions numériques entières.

Instruction identique à **RBOX** mais avec un "rectangle" plein aux bords arrondis. La couleur et le motif de remplissage sont déterminés par l'instruction **DEFFILL**.

Voir également les instructions **BOX**, **DEFFILL**, **BOX** et **RBOX**.

PUT x,y,var\$,mode

x,y,mode : expressions numériques entières ;

var\$: variable alphanumérique.

Cette instruction affiche à l'écran la portion d'écran saisie avec l'instruction **GET** et stockée dans "var\$". Le point de coordonnées (x,y) constitue le coin supérieur gauche du bloc binaire affiché. Le mode a les attributs suivants :

0 = effacer dans la couleur de fond ;

1 = C AND I ;

2 = C AND (NOT I) ;

3 = C (en superposition) ;

4 = (NOT C) AND I ;

5 = I (inchangé) ;

6 = C XOR I ;

7 = C OR I ;

8 = NOT(C OR I) ;

9 = NOT(C XOR I) ;

10 = NOT I (inversion vidéo) ;

11 = C OR (NOT I) ;

12 = NOT C (superposition en inversion vidéo) ;
 13 = (NOT C) OR I ;
 14 = NOT(C AND I) ;
 15 = effacer dans la couleur d'écriture.
 Voir également les instructions GET, SGET et SPUT.

RBOX x1,y1,x2,y2

x1,y1,x2,y2 : expressions numériques entières.

Cette instruction dessine un "rectangle" à coins arrondis entre les points de coordonnées (x1,y1) et (x2,y2) qui constituent respectivement le coin supérieur gauche et le coin inférieur droit du rectangle.

Voir également les instructions BOX, PBOX et PRBOX.

SETCOLOR n,tr,tv,tb

SETCOLOR n,c

n,c,tr,tv,tb : expressions numériques entières.

Cette instruction permet de sélectionner et modifier le registre couleurs numéro "n" (n varie de 0 à 1 en haute résolution, de 0 à 4 en moyenne résolution, de 0 à 15 en basse résolution). Les paramètres "tr", "tv" et "tb" indiquent respectivement les taux de rouge, de vert et de bleu de la nouvelle couleur. Dans la deuxième forme de syntaxe, le paramètre "c" indique la couleur qui correspond au format interne : $c = tr * 256 + tv * 16 + tb$.

Les numéros des couleurs et des registres de l'Atari ST ne correspondent pas. Il s'est produit une erreur lors de l'implantation du VDI. Il existe deux méthodes pour rétablir le bon système de codage : utiliser l'instruction VSETCOLOR ou faire des équivalences à partir de la table suivante :

Table des correspondances des registres de couleurs et des couleurs :

En basse résolution

Couleur	Registre
0	0
2	1
3	2
6	3
4	4
7	5
5	6
8	7
9	8
10	9
11	10
14	11
12	12
15	13
13	14
1	15

En moyenne résolution

Couleur	Registre
0	0
1	3
2	1
3	2

Voir également les instructions **COLOR** et **VSETCOLOR**.

SETDRAW x,y,angle

x,y,angle : expressions numériques entières.

Cette instruction est équivalente à **DRAW "MA",x,y,"TT",angle**.

Voir également les instructions **DRAW** et **DRAW()**.

SGET var\$

var\$: variable alphanumérique.

Cette instruction permet de saisir dans la variable "var\$" l'écran dans son intégralité.

Voir également les instructions **GET**, **PUT** et **SPUT**.

SPRITE var\$,x,y

var\$: variable alphanumérique ;

x,y : expressions numériques entières.

Cette instruction permet d'afficher un lutin dans un rectangle dont la position supérieure gauche a pour coordonnées (x,y). La variable "var\$", composée de 74 caractères, est définie comme suit :

var\$ = MKI\$(coordonnée x du point d'action)
+ MKI\$(coordonnée y du point d'action)
+ MKI\$(0) mode normal ou MKI\$(1) mode XOR
+ MKI\$(couleur du masque) généralement 0
+ MKI\$(couleur du sprite) généralement 1
+ MKI\$(motif binaire 1 du masque)
+ MKI\$(motif binaire 1 du sprite)
...
+ MKI\$(motif binaire 16 du masque)
+ MKI\$(motif binaire 16 du sprite)

Voir également l'instruction **MKI\$**.

SPUT var\$

var\$: variable alphanumérique.

Cette instruction permet d'afficher l'écran saisi avec l'instruction **SGET** et stocké dans la variable "var\$".

Voir également les instructions **GET**, **PUT** et **SGET**.

TEXT x,y [,long],chaîne\$

x,y,long : expressions numériques entières ;

chaîne\$: expression alphanumérique.

Cette instruction affiche le texte "chaîne\$" en mode graphique en commençant

à la position de coordonnées (x,y). Le paramètre facultatif "long" indique la longueur en pixels occupée par la chaîne. Si "long" est positif, la justification se fait sur les caractères (espacement variable des caractères), et si "long" est négatif, la justification se fait sur les mots (espacement variable des mots). Si "long" est omis ou nul, le texte est affiché avec les espacements standards. Voir également l'instruction **DEFTEXT**.

VSETCOLOR n,tr,tv,tb

VSETCOLOR n,c

n,c,tr,tv,tb : expressions entières.

A l'instar de **SETCOLOR**, cette instruction permet de sélectionner et modifier le registre couleurs numéro "n" (n varie de 0 à 1 en haute résolution, de 0 à 4 en moyenne résolution, de 0 à 15 en basse résolution). Les paramètres "tr", "tv" et "tb" indiquent respectivement les taux de rouge, de vert, et de bleu de la nouvelle couleur. Dans la deuxième forme de syntaxe, le paramètre "c" indique la couleur qui correspond au format interne : $c = tr * 256 + tv * 16 + tb$.

Les numéros des couleurs et des registres de l'Atari ST sont remis en ordre avec **VSETCOLOR**, alors que ce n'était pas le cas avec **SETCOLOR**.

Voir également les instructions **COLOR** et **SETCOLOR**.

VSYNC

Cette instruction permet de synchroniser les affichages avec le balayage de l'écran, évitant ainsi les parasites. Le programme est interrompu jusqu'au prochain passage du faisceau, c'est-à-dire que l'image soit entièrement affichée.

PROGRAMMES GRAPHIQUES

147

La syntaxe des instructions est une chose primordiale, mais encore faut-il savoir comment mettre en pratique les différents éléments pour obtenir de belles courbes, des histogrammes ou des camemberts en relief, ou bien encore se construire un petit logiciel de dessin.

Chaque programme est tout d'abord donné dans son intégralité, puis décortiqué afin de vous expliquer comment fonctionnent les diverses instructions. Les programmes fonctionnent dans la mesure du possible dans les trois résolutions de l'Atari ST.

TRACÉ D'HISTOGRAMMES ET DE CAMEMBERTS EN 2D ET 3D

Ce programme donne diverses représentations graphiques à partir de données fictives conservées en DATA.

Il fonctionne dans toutes les résolutions. Toutefois, vous constaterez un problème lors de la représentation du cercle en moyenne résolution provenant de l'instruction CIRCLE elle-même.

Programme principal

Les instructions suivantes sont facultatives; toutefois, elles faciliteront l'écriture du programme. En effet, lorsque les variables sont saisies (mode DEFLIST 2), elles prennent automatiquement le bon suffixe à l'affichage.

DEFBYT "i,j,nombre"	désigne des octets ;
DEFWRD "x,y"	désigne des mots (16 bits) ;
DEFWRD "dx"	désigne des mots (16 bits) ;
DEFWRD "cx,cy"	désigne des mots (16 bits) ;
DEFWRD "lx,ly"	désigne des mots (16 bits) ;
DEFSTR "mo"	désigne une chaîne.

Variables globales :

nombre	: nombre de valeurs à saisir dans les DATA ;
valeur()	: tableau de valeurs ;
mois\$()	: tableau contenant le nom des mois ;
cx&	: facteur de résolution sur l'axe des X ;
cy&	: facteur de résolution sur l'axe des Y ;
nb_coul	: nombre de couleurs (de 0 à 15 maximum) ;
x0&	: abscisse X de l'origine ;
y0&	: ordonnée Y de l'origine ;
lx&	: abscisse maximale par rapport à l'origine x0& ;
ly&	: ordonnée maximale par rapport à l'origine y0& ;
dx&	: intervalle entre deux données selon X.

Procédure resolution

Initialise les variables globales relatives à la résolution d'écran choisie par l'utilisateur.

Procédure repere

Initialise les variables globales du changement de repère et d'échelle.

Procédure trace_rep

Trace le repère selon la résolution d'écran choisie et affiche les mois en abscisse.

Variables locales :

i| et j| : indices de boucles ;
tx& : abscisse du point d'une lettre du texte ;
ty& : ordonnée du point d'une lettre du texte.

Procédure histogramme

Dessine un histogramme en deux dimensions.

Paramètres :

x&() et y&() : tableaux de coordonnées.

Variable locale :

i| : indice de boucle.

Procédure histogramme1

Dessine un histogramme en trois dimensions.

Paramètres :

x&() et y&() : tableaux de coordonnées.

Variables locales :

i| et j| : indices de boucles.

Procédure courbe

Dessine une courbe grâce à l'instruction POLYLINE qui reçoit ses paramètres au moyen de deux tableaux de coordonnées.

Paramètres :

dx& : variation suivant l'axe des X ;

dy& : variation suivant l'axe des Y ;

n| : nombre de points ;

x&() et y&() : tableaux de coordonnées.

Procédure camembert

Dessine un camembert en deux dimensions. Il reçoit comme paramètres un tableau de coordonnées contenant les mesures des secteurs angulaires représentant les données (la somme de ces mesures est égale à trois cent soixante degrés).

Paramètres:

x& : abscisse du centre ;

y& : ordonnée du centre ;

r& : rayon du camembert ;

alpha#() : tableau des mesures des secteurs angulaires.

Variable locale :

i| : indice de boucle.

Procédure camembert1

Dessine un camembert en trois dimensions. Il reçoit comme paramètres un tableau de coordonnées contenant les mesures des secteurs angulaires représentant les données (la somme de ces mesures est égale à trois cent soixante degrés).

Paramètres:

x& : abscisse du centre ;

y& : ordonnée du centre ;

r& : rayon du camembert ;

alpha#() : tableau des mesures des secteurs angulaires.

Variables locales :

i| et j| : indices de boucles.

Procédure normalisation

Elle permet de réduire ou d'agrandir les représentations des données selon la résolution d'écran choisie par l'utilisateur et la valeur de la plus grande des données. De plus, elle initialise les mesures des secteurs angulaires.

Paramètres:

nombre| : nombre de données ;

valeur#() : tableau des valeurs ;

x&() : tableau des abscisses ;

y&() : tableau des ordonnées ;

alpha#() : tableau des mesures des secteurs angulaires.

Variables locales :

i| : indice de boucle ;

fact# : facteur d'échelle ;

maxi# : valeur de la plus grande des données ;

somme# : somme des données ;

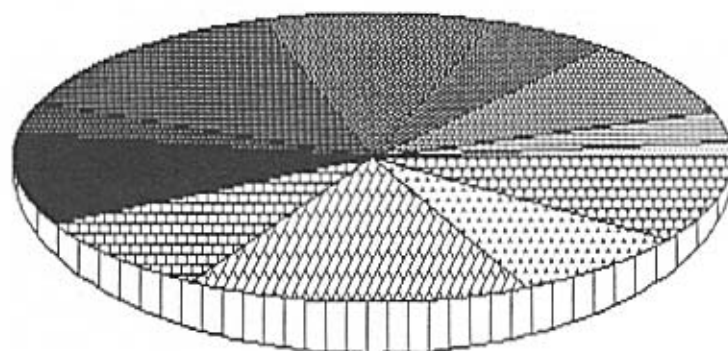
x&() : tableau des abscisses ;

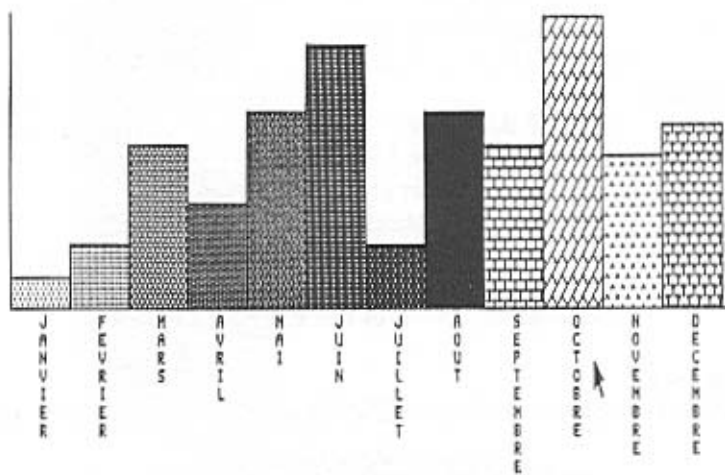
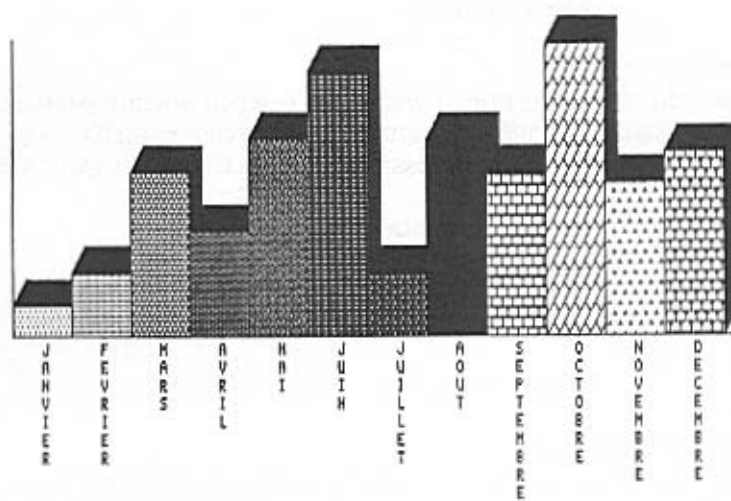
y&() : tableau des ordonnées ;

alpha#() : tableau des mesures des secteurs angulaires.

Fonction getrez

Elle détermine la résolution de l'écran. Elle fait appel à la fonction 4 du XBIOS.





Programme :

```
' -----
'   TRACES DE STATISTIQUES
' -----
' DEFBYT "i,j,nombre"
' DEFWRD "x,y"
' DEFWRD "dx"
' DEFWRD "cx,cy"
' DEFWRD "lx,ly"
' DEFSTR "mo"
'
nombre|=12
DIM valeur(nombre|),mois$(nombre|)
RESTORE donnees
FOR i|=1 TO nombre|
    READ valeur(i|)
NEXT i|
RESTORE mois
FOR i|=1 TO nombre|
    READ mois$(i|)
NEXT i|
repere
normalisation(nombre|,valeur(),x&(),y&(),alpha())
FOR i|=1 TO 12
NEXT i|
CLS
trace_rep
histogramme(x&(),y&())
PAUSE 200
histogramme1(x&(),y&())
PAUSE 200
trace_rep
courbe(-DIV(dx&,2),0,nombre|+1,x&(),y&())
PAUSE 200
CLS
camembert(DIV(lx&,2),-DIV(ly&,2),DIV(ly&,2),alpha())
PAUSE 200
CLS
camembert1 (DIV(lx&,2),-DIV(ly&,2),DIV(lx&,2),DIV(ly&,2),
alpha())
PAUSE 200
' -----
'   PROCEDURES ET FONCTIONS
' -----
PROCEDURE resolution
    SELECT (@getrez)
```

```

CASE 0
  cx&=1
  cy&=1
CASE 1
  cx&=2
  cy&=1
CASE 2
  cx&=2
  cy&=2
ENDSELECT
RETURN
'
PROCEDURE repere
  resolution
  x0&=MUL(30,cx&)
  y0&=MUL(135,cy&)
  lx&=SUB(MUL(310,cx&),x0&)
  ly&=SUB(y0&,MUL(20,cy&))
  dx&=DIV(lx&,nombre|)
  CLIP OFFSET x0&,y0&
RETURN
'
PROCEDURE trace_rep
  LOCAL i|,j|,tx&,ty&
  DRAW 0,0 TO lx&,0
  DRAW 0,0 TO 0,-ly&
  DEFTEXT ,,,4
  FOR i|=1 TO nombre|
    FOR j|=1 TO LEN(mois$(i|))
      tx&=SUB(MUL(dx&,i|),DIV(dx&,2))
      ty&=MUL(j|,7)
      TEXT tx&,ty&,MID$(mois$(i|),j|,1)
    NEXT j|
  NEXT i|
RETURN
'
PROCEDURE histogramme(VAR x&(),y&())
  LOCAL i|
  FOR i|=1 TO nombre|
    DEFFILL ,2,i|
    PBOX x&(i|-1),0,x&(i|),y&(i|)
  NEXT i|
RETURN
'
PROCEDURE histogrammel(VAR x&(),y&())
  LOCAL i|,j|
  FOR i|=1 TO nombre|

```

```

DEFFILL ,2,i|
PBOX x&(i|-1),0,x&(i|),y&(i|)
FOR j|=0 TO 10
    DRAW ADD(x&(i|-1),j|),SUB(y&(i|),j|)
    DRAW TO ADD(x&(i|),j|),SUB(y&(i|),j|)
    DRAW TO ADD(x&(i|),j|),-j|
NEXT j|
NEXT i|
RETURN
,
PROCEDURE courbe(dx&,dy&,n|,VAR x&(),y&())
    x&(0)=-dx&
    y&(0)=-dy&
    POLYLINE n|,x&(),y&() OFFSET dx&,dy&
RETURN
,
PROCEDURE camembert(x&,y&,r&,VAR alpha())
    LOCAL i|
    FOR i|=1 TO DIM?(alpha())-1
        DEFFILL ,2,i|
        PCIRCLE x&,y&,r&,alpha(i|-1),alpha(i|)
    NEXT i|
RETURN
,
PROCEDURE camembert1(x&,y&,rx&,ry&,VAR alpha())
    LOCAL i|
    DEFFILL ,3,10
    PELLIPSE x&,y&+5,rx&,ry&+15,1700,3700
    FOR i|=1 TO nombre|
        DEFFILL ,2,i|
        PELLIPSE x&,y&,rx&,ry&,alpha(i|-1),alpha(i|)
    NEXT i|
RETURN
,
PROCEDURE normalisation(nombre|,VAR valeur(),x&(),y&(),alpha())
    LOCAL i|,fact,maxi,somme
    DIM x&(nombre|),y&(nombre|)
    DIM alpha(nombre|)
    maxi=0
    somme=0
    FOR i|=1 TO nombre|
        maxi=MAX(valeur(i|),maxi)
        ADD somme,valeur(i|)
    NEXT i|
    fact=1y&/maxi
    alpha(0)=0

```

```

FOR i|=1 TO nombre|
  x&(i|)=MUL(i|,dx&)
  y&(i|)=-valeur(i|)*fact
  alpha(i|)=valeur(i|)/somme*3600+alpha(i|-1)
NEXT i|
x&(0)=0
y&(0)=0
RETURN
|
FUNCTION getrez
  RETURN XBIOS(4)
ENDFUNC
|
| -----
|          DONNEES
| -----
|
donnees:
DATA 100,200,500,320,600,800
DATA 200,600,500,890,470,560
mois:
DATA JANVIER,FEVRIER,MARS,AVRIL,MAI,JUIN
DATA JUILLET,AOUT,SEPTEMBRE,OCTOBRE,NOVEMBRE,DECEMBRE

```

UN MINI-PROGRAMME DE DESSIN

Ce programme est une initiation à la construction d'un logiciel de création graphique. Il permet de dessiner, de charger et de sauvegarder des images aux formats Degas et Néochrome.

Fonctionne dans toutes les résolutions. Toutefois, vous constaterez un problème lors de la représentation du cercle en moyenne résolution provenant de l'instruction CIRCLE elle-même.

Programme principal

Les instructions suivantes sont facultatives; toutefois, elles faciliteront l'écriture du programme. En effet, lorsque les variables sont saisies (mode DEFLIST 2), elles prennent automatiquement le bon suffixe à l'affichage.

DEFWRD "a-z" désigne des mots (16 bits);
 DEFBYT "b,n,i" désigne des octets.

Variables globales

men\$()	: tableau contenant le menu déroulant ;
col&()	: tableau des couleurs Néochrome et Degas ;
coul&()	: tableau de sauvegarde des couleurs ;
dessin\$: image écran courante ;

des_back\$: image écran précédente ;
xs& : abscisse de la position de la souris ;
ys& : ordonnée de la position de la souris ;
es| : état des boutons de la souris.

Procédure init

Initialise les couleurs, les images écrans et fait appel à la procédure d'initialisation du menu.

Procédure init_menu

Initialise les items du menu. Elle laisse de la place pour les six accessoires de bureau accessibles depuis le menu déroulant le plus à gauche.

Variable locale

i| : indice de boucle.

Procédure couleur_norme

Initialise les couleurs d'écriture et de fond afin de pouvoir lire clairement son programme en cas de changement de la palette.

Procédure traitement

Permet de faire un branchement sur la procédure concernée à partir de l'item choisi contenu dans MENU(0).

Procédure Info

Donne des informations sur le programme.

Procédure init_dessin

Initialise l'image écran et la forme de la souris.

Procédure retour_menu

Capte l'écran courant et change le curseur souris en une flèche.

Procédure point

Affiche un point tant que le bouton gauche de la souris reste enfoncé. Si aucun bouton n'est pressé, rien ne se passe. On retourne au menu en cliquant sur le bouton droit.

Procédure souris

Détermine la position et l'état de la souris.

Procédure forme_souris

Définit la forme de la souris.

Paramètre :

n| : code de la forme.

Procédure dessin

Selon le paramètre passé, cette procédure dessine une ligne, une X-ligne, une K-ligne, un rectangle ou un cercle. On retourne au menu en cliquant sur le bouton droit.

Paramètre :

n| : code de l'opération à réaliser.

Variables locales :

xi& : abscisse de départ ;

yi& : ordonnée de départ ;

xs& : abscisse courante ;

ys& : ordonnée courante ;

es| : état des boutons de la souris.

Procédure undo

Permet d'annuler une opération en appuyant sur la touche <UNDO>. Cette option n'est disponible que lorsque le menu est actif.

Procédure terminer

Permet de sortir du programme en sauvant, si besoin est, l'image en cours.

Variable locale :

b| : réponse à la boîte d'alerte.

Procédure charger

Permet de charger une image au format Néochrome ou Degas.

Variable locale :

b| : réponse à la boîte d'alerte.

Procédure image_degas

Charge une image au format Degas PI1, PI2 et PI3. Cette procédure vérifie si le fichier choisi est bien une image Degas. Pour des images provenant de Degas Elite, il faut tester une longueur de fichier à 32066 octets et non 32034. Pour plus de sécurité, vous n'avez qu'à changer le test en :

IF (LOF(#1)<>32034 AND LOF(#1)<>32066) OR ...

Variables locales :

b| : réponse à la boîte d'alerte ;

nom\$: chemin ;

choix\$: nom du fichier choisi.

Procédure couleur_degas

Charge dans les registres les couleurs du fichier Degas.

Variable locale :

i| : indice de boucle.

Procédure sauve_degas

Sauvegarde une image au format Degas PI1, PI2 et PI3.

Variables locales :

i| : indice de boucle ;

b| : réponse à la boîte d'alerte ;
nom\$: chemin ;
choix\$: nom du fichier choisi.

Procédure image_neo

Charge une image au format Néochrome. Cette procédure vérifie si le fichier choisi est bien une image Néochrome.

Variables locales :

b| : réponse à la boîte d'alerte ;
nom\$: chemin ;
choix\$: nom du fichier choisi.

Procédure couleur_neo

Charge dans les registres les couleurs du fichier Néochrome.

Variable locale :

i| : indice de boucle.

Procédure sauve_neo

Sauvegarde une image au format Néochrome.

Variables locales :

i| : indice de boucle ;
b| : réponse à la boîte d'alerte ;
nom\$: chemin ;
choix\$: nom du fichier choisi.

Procédure sauvegarde

Propose la sauvegarde de l'image en cours.

Variable locale :

b| : réponse à la boîte d'alerte.

Procédure couleur

Cette procédure modifie la couleur d'écriture.

Variables locales :

i| : indice de boucle ;
xs& : abscisse courante ;
ys& : ordonnée courante ;
es| : état des boutons de la souris.

Procédure sauve_couleur

Sauvegarde dans le tableau coul&() les couleurs actuelles pour une éventuelle modification.

Variable locale :

i| : indice de boucle.

Fonction getcolor

Elle détermine la couleur d'un registre. Elle fait appel à la fonction 7 du XBIOS.

Paramètre :

n| : numéro du registre.

Fonction getrez

Elle détermine la résolution de l'écran. Elle fait appel à la fonction 4 du XBIOS.

Fonction physbase

Elle détermine l'adresse du premier octet de l'écran physique. Elle fait appel à la fonction 2 du XBIOS.

Fonction squ

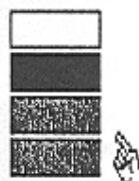
Elle calcule le carré d'un nombre.

Paramètre :

n& : nombre.

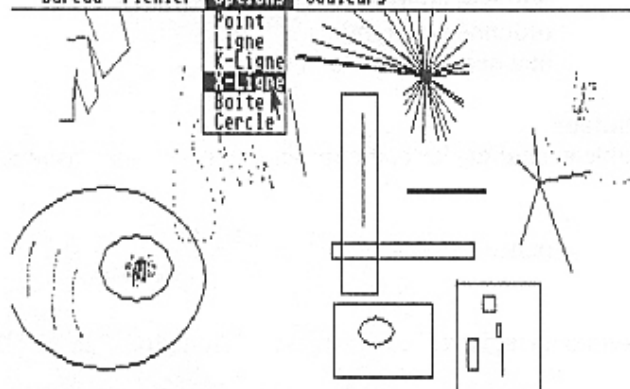
Bureau Fichier Options Couleurs

Point
Ligne
K-Ligne
X-Ligne
Boite
Cercle



Bureau Fichier Options Couleurs

Point
Ligne
K-Ligne
X-Ligne
Boite
Cercle



```

Programme :
' -----
'          DESSIN
' -----
' DEFWRD "a-z"
' DEFBYT "b,n,i"
'
CLEAR
init
MENU men$()
ON MENU GOSUB traitement
ON MENU KEY GOSUB undo
DO
  ON MENU
LOOP
END
' -----
'  PROCEDURES ET FONCTIONS
' -----
PROCEDURE init
  DIM col$(63),coul$(15)
  couleur_norme
  sauve_couleur
  CLS
  SGET dessin$
  SGET des_back$
  init_menu
RETURN
'
PROCEDURE init_menu
  LOCAL i|
  DIM men$(50)
  i|=0
  REPEAT
    READ men$(i|)
    INC i|
  UNTIL men$(i|-1)="*"
  men$(i|-1)=" "
  DATA Bureau, Informations
  DATA -----
  DATA 1,2,3,4,5,6," "
  DATA Fichier, Charger, Sauvegarder, Quitter," "
  DATA Options, Point, Ligne, K-Ligne, X-Ligne
  DATA Boite, Cercle," "
  DATA Couleurs, Couleur," "
  DATA *
RETURN

```

```

'
PROCEDURE couleur_norme
  SETCOLOR 0,1911
  SETCOLOR 15,0
RETURN
'
PROCEDURE traitement
  CLS
  SELECT men$(MENU(0))
  CASE " Inf"
    info
  CASE " Cha"
    charger
  CASE " Sau"
    sauvegarde
  CASE " Qui"
    terminer
  CASE " Poi"
    point
  CASE " Lig"
    dessin(1)
  CASE " K-L"
    dessin(2)
  CASE " X-L"
    dessin(3)
  CASE " Boi"
    dessin(4)
  CASE " Cer"
    dessin(5)
  CASE " Cou"
    couleur
  ENDSELECT
  MENU men$()
RETURN
'
PROCEDURE info
  LOCAL b|
  ALERT 1,"DESSIN",1,"VU",b|
RETURN
'
PROCEDURE init_dessin
  SPUT dessin$
  forme_souris(5)
RETURN
'
PROCEDURE retour_menu
  SGET dessin$

```

```

    forme_souris(0)
RETURN
'
PROCEDURE point
    init_dessin
    DO
        souris
        EXIT IF es|=2
        SGET des_back$
        PLOT xs&,ys&
    LOOP
    retour_menu
RETURN
'
PROCEDURE souris
    REPEAT
        MOUSE xs&,ys&,es|
    UNTIL es|<>0
RETURN
'
PROCEDURE forme_souris(n|)
    DEFMOUSE n|
RETURN
'
PROCEDURE dessin(n|)
    LOCAL xi&,yi&,xs&,ys&,es|
    init_dessin
    DO
        souris
        EXIT IF es|=2
        WHILE MOUSEK=1
        WEND
        xi&=xs&
        yi&=ys&
        SGET des_back$
        i|=1
        IF n|<>5 THEN
            PLOT xs&,ys&
        ENDIF
        REPEAT
            souris
            EXIT IF es|=2
            WHILE MOUSEK=1
            WEND
            IF i|<>1 THEN
                SGET des_back$
            ENDIF
        UNTIL es|=2
    LOOP
ENDPROCEDURE

```

```

i|=0
IF n|=4 THEN
    BOX xi&,yi&,xs&,ys&
ELSE
    IF n|=5 THEN
        CIRCLE xi&,yi&,SQR(@squ(xi&-xs&)+@squ(yi&-ys&))
    ELSE
        DRAW xi&,yi& TO xs&,ys&
    ENDIF
ENDIF
IF n|=2 THEN
    xi&=xs&
    yi&=ys&
ENDIF
UNTIL n|=1 OR n|=4 OR n|=5
LOOP
retour_menu
RETURN
,
PROCEDURE undo
    IF MENU(14)=24832 THEN
        SWAP dessin$,des_back$
        SPUT dessin$
        MENU men$()
    ENDIF
RETURN
,
PROCEDURE terminer
    LOCAL b|
    ALERT 2,"|Voulez-vous sauvegarder
|cette image ?",1,"OUI|NON",b|
    IF b|=1 THEN
        sauvegarde
    ENDIF
    couleur_norme
    STOP
RETURN
,
PROCEDURE charger
    LOCAL b|
    ALERT 2,"|Voulez-vous sauvegarder
|l'image precedente ?",1,"OUI|NON",b|
    IF b|=1 THEN
        sauvegarde
    ENDIF
    ALERT 2,"|Quel est le format
|de l'image a charger ?",2,"NEO|DEGAS",b|

```

```

ON b| GOSUB image_neo,image_degas
sauve_couleur
retour_menu
RETURN
'
PROCEDURE image_degas
LOCAL b|,nom$,choix$
nom$="\*.PI"+CHR$(49+@getrez)
FILESELECT nom$,"",choix$
IF EXIST(choix$) AND choix$<>" " THEN
OPEN "i",#1,choix$
IF LOF(#1)<>32034 OR
RIGHT$(choix$,3)<>"PI"+CHR$(49+@getrez) THEN
ALERT 1,"|Ce n'est pas un fichier |au format DEGAS",1,"VU",b|
ELSE
HIDEM
BLOAD choix$,@physbase-34
couleur_degas
SHOWM
ENDIF
CLOSE #1
ENDIF
RETURN
'
PROCEDURE couleur_degas
LOCAL i|
FOR i|=0 TO 15
col&(i|)=CARD{@physbase-32+MUL(i|,2)}
SETCOLOR i|,col&(i|)
NEXT i|
RETURN
'
PROCEDURE sauve_degas
LOCAL i|,b|,nom$,choix$
HIDEM
SPUT dessin$
FOR i|=0 TO 15
CARD{@physbase-32+MUL(i|,2)}=coul&(i|)
NEXT i|
BYTE{@physbase-33}=@getrez
BYTE{@physbase-34}=0
nom$="\*.PI"+CHR$(49+@getrez)
FILESELECT nom$,"",choix$
IF choix$<>" " THEN
BSAVE choix$,@physbase-34,32034
ENDIF

```

```

    SHOWM
RETURN
'
PROCEDURE image_neo
    LOCAL b|,nom$,choix$
    IF @getrez=0 THEN
        nom$="\*.NEO"
        FILESELECT nom$,"",choix$
        IF EXIST(choix$) THEN
            OPEN "i",#1,choix$
            IF LOF(#1)<>32128 OR RIGHT$(choix$,3)<>"NEO" THEN
                ALERT 1,"|Ce n'est pas un fichier
|au format NEO",1,"VU",b|
            ELSE
                HIDEM
                BLOAD choix$,@physbase-128
                couleur_neo
                SHOWM
            ENDIF
            CLOSE #1
        ENDIF
    ELSE
        ALERT 1,"|Uniquement |en basse resolution !",1,"VU",b|
    ENDIF
RETURN
'
PROCEDURE couleur_neo
    LOCAL i|
    FOR i|=0 TO 15
        col&(i|)=CARD{@physbase-124+MUL(i|,2)}
        SETCOLOR i|,col&(i|)
    NEXT i|
RETURN
'
PROCEDURE sauvegarde
    LOCAL b|
    ALERT 2,"|Dans quel format voulez-vous
|votre image ?",2,"NEO|DEGAS",b|
    ON b| GOSUB sauve_neo,sauve_degas
RETURN
'
PROCEDURE sauve_neo
    LOCAL i|,b|,nom$,choix$
    IF @getrez=0 THEN
        HIDEM
        SPUT dessin$
        ARRAYFILL col&(),0

```



```

FOR i|=2 TO 17
    col&(i|)=coul&(i|-2)
NEXT i|
FOR i|=18 TO 23
    col&(i|)=8224
NEXT i|
col&(22)=11808
FOR i|=0 TO 63
    CARD{@physbase-128+MUL(i|,2)}=col&(i|)
NEXT i|
nom$="\*.NEO"
FILESELECT nom$,"",choix$
IF choix$<>" " THEN
    BSAVE choix$,@physbase-128,32128
ENDIF
SHOWM
ELSE
    ALERT 1,"|Uniquement |en basse resolution !",1,"VU",b|
ENDIF
RETURN
'
PROCEDURE couleur
    LOCAL i|,xs&,ys&,es|
    DEFMOUSE 3
    CLS
    FOR i|=0 TO 16^(1/ADD(1,@getrez))-1
        DEFFILL i|
        BOX 45,8+i|*12,95,18+i|*12
        PBOX 46,9+i|*12,94,17+i|*12
    NEXT i|
    souris
    FOR i|=0 TO 16^(1/ADD(1,@getrez))-1
        IF xs&>45 AND xs&<95 AND ys&>8+i|*12 AND ys&<18+i|*12 THEN
            COLOR i|
        ENDIF
    NEXT i|
    SPUT dessin$
    retour_menu
RETURN
'
PROCEDURE sauve_couleur
    LOCAL i|
    FOR i|=0 TO 15
        coul&(i|)=@getcolor(i|)
    NEXT i|
RETURN
'

```

```

FUNCTION getcolor(n|)
  RETURN XBIOS(7,n|,-1) AND &H777
ENDFUNC
'
FUNCTION getrez
  RETURN XBIOS(4)
ENDFUNC
'
FUNCTION physbase
  RETURN XBIOS(2)
ENDFUNC
'
FUNCTION squ(n&)
  RETURN MUL(n&,n&)
ENDFUNC

```

UNE TORTUE QUI DESSINE DES ARBRES

Nous avons recréé les instructions du langage Logo au moyen de l'instruction DRAW. L'application, illustrant nos propos, dessine un arbre ternaire avec ses fruits.

Fonctionne en basse résolution uniquement (facilement adaptable pour les autres modes).

Programme principal

L'instruction suivante est facultative; toutefois, elle facilitera l'écriture du programme. En effet, lorsque les variables sont saisies (mode DEFLIST 2), elles prennent automatiquement le bon suffixe à l'affichage.

DEFWRD "a-z" désigne des mots (16 bits).

Procédure arbre

Dessine un arbre ternaire (chaque branche se divisant en trois autres).

Paramètres :

n&	: niveau de l'arbre ;
l&	: longueur du tronc ;
a&	: angle de la sous-branche gauche.

Variables locales :

da&	: variation de l'angle ;
c&	: courbure de l'arbre ;
ld&	: longueur de la sous-branche droite ;
lc&	: longueur de la sous-branche centrale ;
lg&	: longueur de la sous-branche gauche.

Procédure tronc

Dessine le tronc.

Paramètres :

n& : niveau de l'arbre ;
l& : longueur du tronc.

Procédure fruit

Dessine un fruit.

Variable locale :

il : indice de boucle.

Procédures simulant le langage LOGO :

forwd

back

turnleft

turnright

penup

pendown

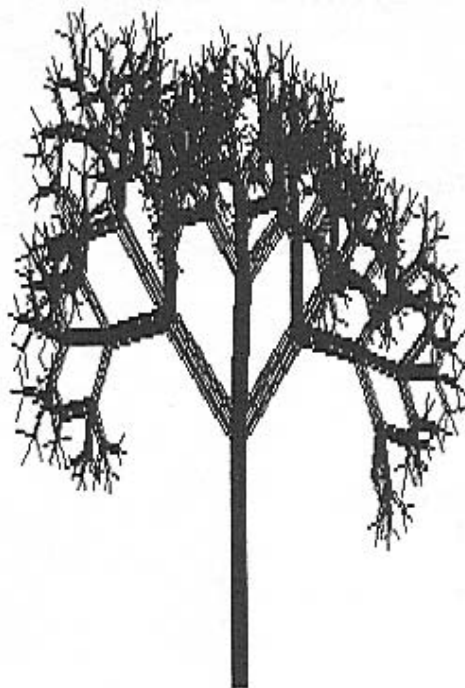
setpencolor

Paramètres :

n& : longueur d'un déplacement ;
a& : angle de rotation.

Procédure couleur

Initialise les couleurs de l'arbre (marron et vert).



Programme :

```

' -----
'               ARBRE
' -----
' DEFWRD "a-z"
'
couleurs
SETDRAW 150,190,0
arbre(7,70,50)
~INP(2)
END

' -----
'   PROCEDURES ET FONCTIONS
' -----

PROCEDURE arbre(n&,l&,a&)
  LOCAL da&,c&,ld&,lc&,lg&
  IF n&>0 AND n&<8 THEN
    IF n&=2 THEN
      fruit
    ENDIF
    setpencolor(ADD(n&,1))
    forwd(l&)
    tronc(DIV(MUL(n&,2),3),l&)
    turnleft(a&)
    ld&=DIV(MUL(l&,10),ADD(RAND(5),13))
    lc&=DIV(MUL(l&,10),ADD(RAND(5),13))
    lg&=DIV(MUL(l&,10),ADD(RAND(5),13))
    da&=RAND(10)-5
    c&=RAND(40)-20
    arbre(PRED(n&),ld&,SUB(a&,da&))
    turnright(SUB(a&,c&))
    arbre(PRED(n&),lc&,SUB(a&,da&))
    turnright(ADD(a&,c&))
    arbre(PRED(n&),lg&,SUB(a&,da&))
    turnleft(a&)
    penup
    back(l&)
    pendown
  ENDIF
RETURN
'

PROCEDURE tronc(n&,l&)
  turnleft(90)
  forwd(n&)
  turnleft(90)
  forwd(l&)

```

```

    turnleft(90)
    forwd(MUL(n&,2))
    turnleft(90)
    forwd(1&)
    turnleft(90)
    forwd(n&)
    turnright(90)
    IF n&>0 THEN
        tronc(n&-1,1&)
    ENDIF
RETURN
'
PROCEDURE fruit
    LOCAL i|
    setpencolor(9)
    FOR i|=1 TO 8
        forwd(1)
        turnleft(45)
    NEXT i|
RETURN
'
PROCEDURE forwd(n&)
    DRAW "fd",n&
RETURN
'
PROCEDURE back(n&)
    DRAW "bk",n&
RETURN
'
PROCEDURE turnleft(a&)
    DRAW "lt",a&
RETURN
'
PROCEDURE turnright(a&)
    DRAW "rt",a&
RETURN
'
PROCEDURE penup
    DRAW "pu"
RETURN
'
PROCEDURE pendown
    DRAW "pd"
RETURN
'
PROCEDURE setpencolor(n&)
    DRAW "co",n&

```

```

RETURN
'
PROCEDURE couleurs
  VSETCOLOR 2,0,6,0
  VSETCOLOR 3,1,5,0
  VSETCOLOR 4,2,4,0
  VSETCOLOR 5,1,4,2
  VSETCOLOR 6,4,3,1
  VSETCOLOR 7,4,3,2
  VSETCOLOR 8,4,3,3
  VSETCOLOR 9,7,1,0
RETURN

```

UN LUTIN AU BOUT DE LA SOURIS

Ou comment déplacer un cannibale avec la souris. Ce petit programme illustre la manipulation de l'instruction SPRITE. Des petits problèmes apparaissent lors de la collision de lutins. Nous vous proposons une solution pour éviter ce genre de désagréments.

Fonctionne dans toutes les résolutions.

Programme principal

Il gère le déplacement du lutin avec la souris.

Les instructions suivantes sont facultatives; toutefois, elles faciliteront l'écriture du programme. En effet, lorsque les variables sont saisies (mode DEFLIST 2), elles prennent automatiquement le bon suffixe à l'affichage.

```

DEFBYT "i"           désigne des octets ;
DEFWRD "xs,ys,dx,dy" désigne des mots (16 bits) ;
DEFWRD "lx,ly"       désigne des mots (16 bits) ;

```

Variables globales :

i	: indice de boucle ;
xs&	: abscisse de la souris ;
ys&	: ordonnée de la souris ;
dx&	: déplacement sur X ;
dy&	: déplacement sur Y ;
lx&	: abscisse maximale de l'écran ;
ly&	: ordonnée maximale de l'écran.

Procédure resolution

Détermine la résolution de l'écran et définit les facteurs en conséquence.

Procédure init_sprites

Initialise les lutins.

Fonction getrez

Renvoie la résolution de l'écran au moyen de la fonction 4 du XBIOS.



Programme :

```

| -----
|          SPRITES
| -----
| DEFBYT "i"
| DEFWRD "xs,ys,dx,dy"
| DEFWRD "lx,ly"
|
init_sprites
resolution
HIDEM
xs&=0
ys&=0
WHILE MOUSEK=0
  dx&=MOUSEX-xs&
  dy&=MOUSEY-ys&
  ADD xs&,dx&
  ADD ys&,dy&
  IF xs&>SUB(lx&,16) THEN
    xs&=SUB(lx&,16)
  ENDIF
  IF ys&>SUB(ly&,48) THEN
    ys&=SUB(ly&,48)
  ENDIF
  SETMOUSE xs&,ys&
  IF dx&<>0 OR dy&<>0 THEN
    CARD(V:africa1$+8)=ADD(RAND(nb_coul),1)
    CARD(V:africa2$+8)=ADD(RAND(nb_coul),1)
    CARD(V:africa3$+8)=ADD(RAND(nb_coul),1)
    IF dy&>0 THEN
      SPRITE africa3$,xs&,ys&+32
      SPRITE africa2$,xs&,ys&+16
    
```

```

        SPRITE africal$,xs&,ys&
    ELSE
        SPRITE africal$,xs&,ys&
        SPRITE africa2$,xs&,ys&+16
        SPRITE africa3$,xs&,ys&+32
    ENDIF
ENDIF
WEND
SHOWM
END
' -----
'           PROCEDURES ET FONCTIONS
' -----
PROCEDURE resolution
    SELECT (@getrez)
    CASE 0
        lx&=320
        ly&=200
        nb_coul|=15
    CASE 1
        lx&=640
        ly&=200
        nb_coul|=3
    CASE 2
        lx&=640
        ly&=400
        nb_coul|=1
    ENDSELECT
RETURN
'
PROCEDURE init_sprites
    RESTORE africal
    africal$=MKI$(0)+MKI$(0)+MKI$(0)
    africal$=africal$+MKI$(0)+MKI$(1)
    FOR i|=1 TO 16
        READ avant%,arriere%
        africal$=africal$+MKI$(arriere%)+MKI$(avant%)
    NEXT i|
    RESTORE africa2
    africa2$=MKI$(0)+MKI$(0)+MKI$(0)
    africa2$=africa2$+MKI$(1)+MKI$(2)
    FOR i|=1 TO 16
        READ avant%,arriere%
        africa2$=africa2$+MKI$(arriere%)+MKI$(avant%)
    NEXT i|
    RESTORE africa3
    africa3$=MKI$(0)+MKI$(0)+MKI$(0)

```



```

africa3$=africa3$+MKI$(0)+MKI$(1)
FOR i|=1 TO 16
  READ avant%,arriere%
  africa3$=africa3$+MKI$(arriere%)+MKI$(avant%)
NEXT i|
RETURN
'
FUNCTION getrez
  RETURN XBIOS(4)
ENDFUNC
' -----
'          DONNEES DES SPRITES
' -----
africa1:
DATA 768,0,4032,0,8160,0,53196,0
DATA 32760,0,53196,0,8160,0,15600,0
DATA 24600,0,16392,0,23784,0,17160,0
DATA 18504,0,20424,0,8208,0,6240,0
africa2:
DATA 1156,0,1166,0,1166,0,2143,0
DATA 6244,0,4132,0,8212,0,8212,0
DATA 16396,0,18508,0,22636,0,39012,0
DATA 38996,0,26700,0,4132,0,4132,0
africa3:
DATA 8164,0,8164,0,8164,0,8164,0
DATA 8164,0,5284,0,5284,0,5284,0
DATA 5284,0,5284,0,5284,0,5284,0
DATA 5284,0,5284,0,30840,0,63612,0

```

DES ROUTINES À INTÉGRER DANS VOS PROGRAMMES

Remettre à jour la palette de couleurs

Lors des manipulations sur les couleurs, l'utilisateur perd sa palette originale (celle qui s'installe à la mise en route). Voilà pourquoi il est intéressant de se créer une palette standard qui gardera les bonnes couleurs aux bons registres. Pour cela, nous allons utiliser l'instruction VSETCOLOR (non documentée dans le GFA 3.0) qui est similaire à SETCOLOR, sauf que l'on n'a pas à se soucier de la table de transcodage entre numéros des couleurs et numéros des registres.

Voici les couleurs de la palette standard :

Numéro	Couleur	Codage
0	BLANC	7,7,7
1	NOIR	0,0,0
2	ROUGE	7,0,0
3	VERT	0,6,0
4	BLEU	0,0,7
5	BLEU MARINE	0,0,5
6	BRUN	5,2,0
7	VERT FONCE	0,5,0
8	GRIS	5,5,5
9	GRIS FONCE	2,2,2
10	BLEU CLAIR	0,7,7
11	BLEU VERT	0,5,5
12	VIOLET	7,0,7
13	VIOLET FONCE	5,0,5
14	JAUNE FONCE	5,5,0
15	JAUNE CLAIR	7,7,0

PROCEDURE couleur_reset

```

VSETCOLOR 0,7,7,7
VSETCOLOR 1,0,0,0
VSETCOLOR 2,7,0,0
VSETCOLOR 3,0,6,0
VSETCOLOR 4,0,0,7
VSETCOLOR 5,0,0,5
VSETCOLOR 6,5,2,0
VSETCOLOR 7,0,5,0
VSETCOLOR 8,5,5,5
VSETCOLOR 9,2,2,2
VSETCOLOR 10,0,7,7
VSETCOLOR 11,0,5,5
VSETCOLOR 12,7,0,7
VSETCOLOR 13,5,0,5
VSETCOLOR 14,5,5,0
VSETCOLOR 15,7,7,0
RETURN

```

Quelle est la résolution de l'écran ?

Voici comment déterminer la résolution de l'écran au moyen de la fonction 4 du XBIOS (getrez).

```

FUNCTION getrez
  RETURN XBIOS(4)
ENDFUNC

```

Quelle est la couleur d'un registre ?

Voici comment déterminer la couleur d'un registre au moyen de la fonction 7 du XBIOS (setcolor).

```
FUNCTION getcolor(nl)
  RETURN XBIOS(7,nl,-1) AND &H777
ENDFUNC
```

Quelle est l'adresse de l'écran physique ?

Voici comment déterminer l'adresse du premier octet de l'écran physique au moyen de la fonction 2 du XBIOS (physbase).

```
FUNCTION physbase
  RETURN XBIOS(2)
ENDFUNC
```

Quelle est l'adresse de l'écran virtuel ?

Voici comment déterminer l'adresse du premier octet de l'écran physique au moyen de la fonction 3 du XBIOS (logbase).

```
FUNCTION logbase
  RETURN XBIOS(3)
ENDFUNC
```


LIGNE A

179

Afin de simplifier la vie du programmeur, la ligne A offre des fonctions graphiques primaires plus compliquées, mais également beaucoup plus rapides que les fonctions graphiques GEM. Le nom provient du branchement effectué par le MC 68000 au vecteur d'exception \$10 lors de la rencontre de codes dont les 4 bits de poids fort sont 1010, soit A en hexadécimal.

Voici la liste des instructions traitées dans ce chapitre :

ACHAR	ARECT	L~A
ACLIP	ATEXT	PSET
ALINE	BIBLT	PTST
APOLY	HLINE	

ACHAR code,x,y,fonte,style,angle

code,x,y,fonte,style,angle : expressions numériques entières.

Cette instruction édite à la position (x,y) spécifiée le caractère dont le code ASCII est le paramètre code.

fonte :

0 : 6x6 ;

1 : 8x8 ;

2 : 8x16 ;

au-delà de 3 (header de fontes).

style : 0 à 31 ;

angle : 0, 900, 1800 ou 2700.

ACLIP flag,x0,y0,x1,y1

flag : expression numérique booléenne ;

x0,y0,x1,y1 : expressions numériques entières.

Cette instruction limite les tracés des fonctions graphiques de la LINE-A (clipping) à un rectangle dont le coin supérieur gauche a pour coordonnées (x0,y0) et le coin inférieur droit a pour coordonnées (x1,y1). Lorsque flag=0 : il y a désactivation de la zone de limitation. Attention : les instructions ALINE, BIBLT, HLINE, PSET et PTST ne sont pas concernés par cette limitation !

ALINE x1,y1,x2,y2,couleur,masque,mode

x1,y1,x2,y2,couleur,masque,mode : expressions numériques entières.

Cette instruction trace une ligne entre les points de coordonnées (x1,y1) et (x2,y2).

mode :

1 : remplacement ;

2 : transparent ;

3 : inversion ;

4 : transparent inverse.

APOLY adr1,n1,y1 TO y2,couleur,mode,adr2,n2

adr1,n1,y1,2,couleur,mode,adr2,n2 : expressions numériques entières.

Cette instruction trace un polygone tramé (sans ligne de contour). Le tableau de coordonnées pointé par adr1 est structuré comme une liste des couples de coordonnées des points sommets du polygone. Le remplissage de trame est compris entre les deux limites horizontales y1 et y2.

Le paramètre couleur indique la couleur entre 0 et 15.

Le mode graphique ne correspond pas à celui de GRAPHMODE :

- 0 : remplacement ;
- 1 : transparent ;
- 2 : XOR (OU exclusif) ;
- 3 : transparent inverse.

l'adresse adr2 pointe sur un tableau qui contient des mots qui codent le motif graphique. Le paramètre n2 contient le nombre de motifs à prendre avant de recommencer avec le premier motif (n2=0).

ARECT x1,y1,x2,y2,couleur,mode,adr,n

x1,y1,x2,y2,couleur,mode,adr,n : expressions numériques entières.

Cette instruction est équivalente à PBOX, elle trace un rectangle plein entre les points de coordonnées (x1,y1) et (x2,y2). Le paramètre couleur indique la couleur entre 0 et 15.

Le mode graphique ne correspond pas à celui de GRAPHMODE :

- 0 : remplacement ;
- 1 : transparent ;
- 2 : XOR (OU exclusif) ;
- 3 : transparent inverse.

l'adresse adr pointe sur un tableau qui contient des mots qui codent le motif graphique. Le paramètre n contient le nombre de motifs à prendre avant de recommencer avec le premier motif (n=0).

ATEXT x,y,fonte,ch\$

ch\$: expression alphanumérique ;

x,y,fonte : expressions numériques entières.

Cette instruction édite la chaîne de caractères ch\$ à l'écran aux points de coordonnées (x,y).

fonte :

- 0 : 6x6 ;
- 1 : 8x8 ;
- 2 : 8x16 ;

au-delà de 3 (header de fontes).

BITBLT bloc-source(),bloc-objet(),param()

BITBLT adr

BITBLT tab()

Cette instruction permet la copie de bloc paramétrée (VDI).

Paramètres :

bloc-source() et bloc-objet() :

- (0) : adresse du bloc ;
- (1) : largeur du bloc (multiple de 16)
exprimée en pixels ;

- (2) : hauteur du bloc (pixels) ;
- (3) : largeur du bloc en nombre de mots
16 bits (largeur pixels/16) ;
- (4) : 0 ;
- (5) : nombre de plans de bits : 1 en haute
résolution, 2 en moyenne, 4 en basse.

- param(0) : abscisse (X) du coin supérieur gauche
(rectangle source) ;
- (1) : ordonnée (Y) du coin supérieur gauche
(rectangle source) ;
 - (2) : abscisse (X) du coin inférieur droit
(rectangle source) ;
 - (3) : ordonnée (Y) du coin inférieur droit
(rectangle source) ;
 - (4) : abscisse (X) du coin supérieur gauche
(rectangle objet) ;
 - (5) : ordonnée (Y) du coin supérieur gauche
(rectangle objet) ;
 - (6) : abscisse (X) du coin inférieur droit
(rectangle objet) ;
 - (7) : ordonnée (Y) du coin inférieur droit
(rectangle objet) ;
 - (8) : mode graphique: entre 0 et 15 (voir
l'instruction d'édition de bloc : PUT).

En GFA Basic, l'instruction BITBLT avec un seul paramètre est un appel de la LINE-A, qui permet la copie de bloc paramétrée. Le paramètre adr est la première adresse d'une zone de 76 octets. Le tableau tab() est un tableau de taille minimale égale à vingt-trois mots longs (32 bits). La table ci-dessous indique donc deux indices : suivant i-8 (en nombre d'octets au-delà de adr), ou suivant i-32 (indice du tableau t) :

i-8	i-32	
00	00	Largeur du bloc (pixels) ;
02	01	Hauteur du bloc (pixels) ;
04	02	Nombre de plans de couleur (destination) ;
06	03	Couleur de premier plan destination ;
08	04	Couleur de l'arrière-plan ;
10	05	Mode graphique (cf PUT) ;
14	06	Origine X du bloc source ;
16	07	Origine Y du bloc source ;
18	08	Adresse de base du bloc source.
22	09	Décalage (offset) du mot suivant dans la ligne ;
24	10	Décalage vers la ligne suivante dans le plan (bloc source) ;

26	11	Décalage vers prochain niveau de couleur ;
28	12	Minimum horizontal (destination) ;
30	13	Minimum vertical (destination) ;
32	14	Adresse du bloc destination ;
36	15	Décalage vers mot suivant dans la ligne (destination) ;
38	16	Décalage vers ligne suivante dans le plan (destination) ;
40	17	Décalage vers le prochain niveau de couleur ;
42	18	Adresse de la zone destination ;
46	19	Décalage vers prochaine ligne du masque ;
48	20	Décalage vers la couleur suivante du motif de remplissage ;
50	21	Facteur de répétition du motif (n-motifs dans l'instruction HLINE) ;
52	22	Les vingt-quatre octets qui suivent doivent être nuls, c'est une zone de travail du blitter.

HLINE x1,y,x2,couleur,mode,adr,n

x1,y,x2,couleur,mode,adr,n : expressions numériques entières.

Cette instruction trace une ligne horizontale entre les points de coordonnées (x1,y) et (x2,y).

Le paramètre couleur indique la couleur entre 0 et 15.

Le mode graphique ne correspond pas à celui de GRAPHMODE :

- 0 : remplacement ;
- 1 : transparent ;
- 2 : XOR (OU exclusif) ;
- 3 : transparent inverse.

l'adresse adr pointe sur un tableau qui contient des mots qui codent le motif graphique. Le paramètre n contient le nombre de motifs à prendre avant de recommencer avec le premier motif (n=0).

L~A

Cette fonction renvoie l'adresse de base des variables de la ligne A.

PSET x,y,couleur

x,y,couleur : expressions numériques entières.

Cette instruction est équivalente à PLOT.

PTST(x,y)

x,y : expressions numériques entières.

Cette instruction est équivalente à POINT.

**GESTION
DES FENETRES,
DES MENUS
ET DE LA SOURIS**

185

Pour manipuler les fenêtres, les boîtes d'alerte, la souris, les menus déroulants, le GFA Basic 3.0 met à la disposition du programmeur une panoplie complète d'instructions et de fonctions. Pratiques et faciles à mettre en oeuvre, elles permettent d'obtenir rapidement une présentation des applications plus soignée, plus professionnelle.

Voici la liste des instructions et des fonctions traitées dans ce chapitre :

ALERT	MENU KILL	ON MENU IBOX
CLEARW	MENU OFF	ON MENU KEY
CLOSEW	MENU()	ON MENU MESSAGE
CLS	MOUSE	ON MENU OBOX
CRSCOL	MOUSEK	OPENW
CRSLIN	MOUSEX	SHOWM
DEFMOUSE	MOUSEY	SETMOUSE
FULLW	MENU	TITLEW
HIDEM	ON MENU GOSUB	TOPW
INFOW	ON MENU BUTTON	WINTAB
MENU		

On parlera aussi de deux fonctions permettant la communication entre GEM et le GFA Basic : W_HAND et W_INDEX.

ALERT a,ch1\$,b,ch2\$,variable

a,b : expressions numériques entières ;

ch1\$,ch2\$: expressions alphanumériques ;

variable : variable numérique entière.

Cette instruction affiche une boîte d'alarme au centre de l'écran. Le paramètre "a" désigne le symbole affiché dans le ON panneau d'alerte (en haut à gauche).

a = 0 pas de symbole ;

a = 1 !

a = 2 ?

a = 3 STOP.

La chaîne ch1\$ contient le texte principal. Les lignes (quatre au maximum) sont séparées par le caractère "|", ch2\$ contient le texte relatif à chaque bouton de choix (trois au maximum), également séparés par le caractère "|", b indique lequel des trois boutons est (surligné) pris par défaut si l'on tape sur la touche <RETURN> (ou <ENTER>). La variable contiendra le numéro du bouton sélectionné (b=1, 2, ou 3).

Par exemple :

Alert 1,"Uniquement en basse résolution",1,"VU",B%

**CLEARW n**

n : expression numérique entière.

Efface le contenu de la fenêtre numéro n.

CLOSEW n

n : expression numérique entière.

Cette instruction ferme la fenêtre numéro n.

CLS

Cette instruction efface l'écran.

CRSCOL

Cette fonction renvoie la colonne du curseur texte.

CRSLIN

Cette fonction renvoie la ligne du curseur texte.

DEFMOUSE n**DEFMOUSE var\$**

n : expression numérique entière ;

var\$: expression alphanumérique.

Cette instruction définit la forme de la souris :

n = 0 Flèche ;

n = 1 X allongé ;

n = 2 Abeille ;

n = 3 Main avec doigt pointé ;

n = 4 Main plate ;

n = 5 Réticule fin ;

n = 6 Réticule épais ;

n = 7 Réticule avec contour.

Si l'on désire donner soi-même, le motif de la souris il suffit de définir var\$ ainsi :

```
var$ = MKIS(coordonnée x du point d'action)
      + MKIS(coordonnée y du point d'action)
      + MKIS(1)
      + MKIS(couleur du masque) généralement 0
      + MKIS(couleur du curseur) généralement 1
```

+ M\$ M\$ étant composé de 16 MKI\$
 + C\$ C\$ étant composé de 16 MKI\$
 voir MKI\$ pour plus de précision

FULLW n

n :expression numérique entière.

Cette instruction agrandit la fenêtre numéro n à la taille de l'écran tout entier.

HIDEM

Cette instruction cache le curseur de la souris.

INFOW n,"ligne"

n :expression numérique entière.

Cette instruction place ou remplace la ligne d'information (située sous le titre) de la fenêtre n. INFOW n,"" doit précéder OPENW, si l'on désire une fenêtre sans ligne d'information.

MENU tableau\$()

tableau\$() : tableau alphanumérique.

Cette instruction affiche et active le menu contenu dans tableau\$(), tableau alphanumérique dimensionné avec une valeur au moins égale au nombre d'options. Ce dernier définit à lui seul l'ensemble des paramètres utiles à la création du menu ; ses titres et ses différentes options. Ses éléments sont initialisés comme suit :

tableau\$(0) = titre 1 (principal) du menu ;

tableau\$(1) = réservé en général à la ligne d'information ;

tableau\$(2) = série de traits d'union désactivant la ligne qui est destinée à séparer les accessoires ;

tableau\$(3) = chaîne muette non vide réservant de la place pour l'accessoire numéro 1 ;

tableau\$(4) = chaîne muette non vide réservant de la place pour l'accessoire numéro 2 ;

tableau\$(5) = chaîne muette non vide réservant de la place pour l'accessoire numéro 3 ;

tableau\$(6) = chaîne muette non vide réservant de la place pour l'accessoire numéro 4 ;

tableau\$(7) = chaîne muette non vide réservant de la place pour l'accessoire numéro 5 ;

tableau\$(8) = chaîne muette non vide réservant de la place pour l'accessoire numéro 6 ;

tableau\$(9) = chaîne vide pour indiquer la fin du titre 1 ;

tableau\$(10) = titre 2 du menu (10 caractères maximum) ;

tableau\$(11) = option 1 du titre 2 ;

tableau\$(12) = option 2 du titre 2 ;

tableau\$(13) = option 3 du titre 2 ;

...

tableau\$(10+n) = option n du titre 2 ;

tableau\$(10+n+1) = chaîne vide pour indiquer la fin du titre 2 ;

...

tableau\$(x) = chaîne vide pour indiquer la fin du titre m ;

tableau\$(x+1) = chaîne vide pour indiquer la fin du menu.

La structure employée pour le titre 2 doit l'être également pour les suivants (tableau(10) à tableau(10+n+1)). Les options commençant par un trait d'union sont inactivées et affichées en grisé. Deux chaînes vides terminent le menu.

MENU KILL

Cette instruction désactive le menu déroulant actuellement fonctionnel sans l'effacer. On peut à la suite de cette instruction activer un nouveau menu déroulant avec les instructions MENU tableau\$() et ON MENU GOSUB proc.

MENU OFF

Cette instruction réinitialise le menu ; les titres éventuellement affichés en inversion vidéo sont à nouveau affichés en clair, il est conseillé de placer cette instruction en fin de la procédure de traitement du menu déroulant afin de retrouver celui-ci prêt à un nouvel emploi.

MENU(indice)

indice : expression numérique entière.

Retourne les renseignements nécessaires au traitement d'un événement, ceux-ci sont renvoyés dans le tableau MENU(), les indices variant de 0 à 15 :

MENU(1) identifie l'événement qui s'est produit. Selon les valeurs qu'il prend, on trouvera dans les indices suivants les compléments nécessaires au traitement de l'événement. La valeur appelée handle identifie le gestionnaire d'une application, en l'occurrence une fenêtre :

MENU(1) = 10 choix du menu déroulant ;

MENU(0) = index de l'option choisie dans le tableau de définition ;

MENU(4) = numéro du titre ;

MENU(5) = numéro de l'option ;

MENU(1) = 20 redessine une partie de la fenêtre ;

MENU(4) = handle ;

MENU(5) = coordonnée x de la fenêtre ;

MENU(6) = coordonnée y de la fenêtre ;

MENU(7) = largeur de la fenêtre ;

MENU(8) = hauteur de la fenêtre ;

MENU(1) = 21 clic sur la fenêtre ;

MENU(4) = handle ;

MENU(1) = 22 clic sur la case de fermeture ;

MENU(4) = handle ;

MENU(1) = 23 clic sur la case d'agrandissement maximal ;

MENU(4) = handle ;

MENU(1) = 24 clic sur une des quatre flèches ou une des deux barres grises, remonte d'une ligne le contenu d'une fenêtre ;

MENU(5) = indice du scrolling choisi :

0 = page entière vers le haut ;

1 = page entière vers le bas ;
 2 = une ligne vers le haut ;
 3 = une ligne vers le bas ;
 4 = page entière vers la gauche ;
 5 = page entière vers la droite ;
 6 = un cran vers la gauche ;
 7 = un cran vers la droite ;
 MENU(1) = 25 clic sur l'ascenseur horizontal ;
 MENU(4) = handle ;
 MENU(5) = position relative de l'ascenseur (0-1000) ;
 MENU(1) = 26 clic sur l'ascenseur vertical ;
 MENU(4) = handle ;
 MENU(5) = position relative de l'ascenseur (0-1000) ;
 MENU(1) = 27 clic sur la case de changement de taille ;
 MENU(4) = handle ;
 MENU(5) = coordonnée x de la fenêtre ;
 MENU(6) = coordonnée y de la fenêtre ;
 MENU(7) = nouvelle largeur de la fenêtre ;
 MENU(8) = nouvelle hauteur de la fenêtre ;
 MENU(1) = 28 clic sur la barre de déplacement ;
 MENU(4) = handle ;
 MENU(5) = nouvelle coordonnée x de la fenêtre ;
 MENU(6) = nouvelle coordonnée y de la fenêtre ;
 MENU(7) = largeur de la fenêtre ;
 MENU(8) = hauteur de la fenêtre ;
 MENU(1) = 29 activation d'une nouvelle fenêtre (appartenant à un accessoire) ;
 MENU(4) = handle ;
 MENU(1) = 40 sélection d'un accessoire ;
 MENU(5) = identificateur du menu ;
 MENU(1) = 41 désactivation d'un accessoire ;
 MENU(4) = identificateur du menu ;
 MENU(9) contient le drapeau de l'événement, celui-ci est codé en une série de bits dont chacun représente une catégorie d'événement :
 bit 1 = 1 appui des touches du clavier ;
 bit 2 = 1 appui des touches de la souris ;
 bit 3 = 1 appui de la touche 1 de la souris ;
 bit 4 = 1 appui de la touche 2 de la souris ;
 bit 5 = 1 message GEM (par exemple clic sur la case d'agrandissement) ;
 bit 6 = 1 timer ;
 MENU(10) abscisse de la souris ;
 MENU(11) ordonnée de la souris ;
 MENU(12) touches de la souris :
 1 = touche gauche ;
 2 = touche droite ;
 3 = les deux touches ;
 MENU(13) état des touches de contrôle ;

1 = <SHIFT droit> ;
 2 = <SHIFT gauche> ;
 4 = <CONTROL> ;
 8 = <ALTERNATE> ;

si plusieurs touches sont simultanément appuyées, on obtient l'addition des codes élémentaires, par exemple <SHIFT droit>+<ALTERNATE> = 9 ;

MENU(14) code clavier de la touche appuyée ;

octets poids fort = code SCAN ;

octets poids faible = code ASCII ;

MENU(15) nombre de clics associés à l'événement ;

MOUSE x,y,k

x,y,k : expressions numériques entières.

Cette instruction place dans les variables x, y les coordonnées de la souris et dans la variable k l'état instantané des touches de la souris :

k = 0 aucune touche n'est enfoncée ;
 k = 1 la touche droite est enfoncée ;
 k = 2 la touche gauche est enfoncée ;
 k = 3 les deux touches sont enfoncées.

MOUSEK

Contient l'état instantané des touches de la souris (voir MOUSE).

MOUSEX

Contient l'abscisse de la souris

MOUSEY

Contient l'ordonnée de la souris.

ON MENU

Cette instruction provoque, lors d'un événement, le branchement aux procédures indiquées par les instructions ON MENU, ON MENU IBOX GOSUB, ON MENU OBOX GOSUB, ON MENU MESSAGE GOSUB, ON MENU BUTTON GOSUB et ON MENU KEY GOSUB.

ON MENU GOSUB procédure

Cette séquence d'instructions définit la procédure à appeler lorsque l'une des options du menu déroulant a été sélectionnée. La procédure déterminera le traitement correspondant grâce à MENU(0) (voir MENU(indice)).

ON MENU BUTTON a,b,c GOSUB procédure

a,b,c : expressions numériques entières.

Cette séquence d'instructions définit la procédure à appeler lorsque la souris change d'état (voir MENU(indice)).

Le paramètre "a" indique le nombre de clics maximal à scruter, le paramètre "b" désigne le ou les boutons à examiner :

b = 0 aucun ;

b = 1 bouton gauche ;

b = 2 bouton droit ;

b = 3 les deux boutons.

Le paramètre "c" indique l'état du ou des boutons (0 = non enfoncé, 1 = enfoncé).

ON MENU IBOX i,x,y,l,h GOSUB procédure

i,x,y,l,h : expressions numériques entières.

Cette séquence d'instructions définit la procédure à appeler lorsque la souris entre à l'intérieur d'un rectangle défini par son coin supérieur gauche (coordonnées x et y), sa longueur et sa hauteur (voir MENU(indice)).

ON MENU KEY GOSUB procédure

Cette séquence d'instructions définit la procédure à appeler lorsque l'on agit sur le clavier. La variable MENU(14) contient l'ensemble des codes de la touche pressée (voir MENU(indice)).

ON MENU MESSAGE GOSUB procédure

Cette séquence d'instructions définit la procédure à appeler lorsque l'on agit sur une fenêtre, par exemple si l'on clique sur le bouton d'agrandissement (voir MENU(indice)).

ON MENU OBOX i,x,y,l,h GOSUB procédure

i,x,y,l,h : expressions numériques entières.

Cette séquence d'instructions définit la procédure à appeler lorsque la souris entre à l'extérieur d'un rectangle défini par son coin supérieur gauche (coordonnées x et y), sa longueur et sa hauteur (voir MENU(indice)).

OPENW n[,x,y]

n,x,y : expressions numériques entières.

Cette instruction ouvre la fenêtre numéro n. Si les paramètres optionnels x et y sont spécifiés, ils correspondent aux coordonnées du point d'intersection des quatre fenêtres possibles.

SHOWM

Cette instruction visualise le curseur de la souris.

SETMOUSE x,y[,k]

x,y,k : expressions numériques entières.

Cette instruction place le curseur de la souris au point de coordonnées (x,y). Le paramètre "k" permet de simuler une pression sur un bouton de cette manière :

k = 0 aucune touche n'est enfoncée ;

k = 1 la touche droite est enfoncée ;

k = 2 la touche gauche est enfoncée ;

k = 3 les deux touches sont enfoncées.

TITLEW n,"titre"

n : expression numérique entière.

Cette instruction donne ou redonne un titre à la fenêtre n. TITLEW n,"" doit précéder OPENW, si l'on désire une fenêtre sans titre.

TOPW #n

n : expression numérique entière.

Cette instruction active la fenêtre numéro n.

WINTAB**WINTAB(i,j)**

i,j : expressions numériques entières.

Toutes les routines, utilisant les fenêtres, nécessitent l'emploi d'une table appelée WINTAB composée de mots de 16 bits. Cette dernière peut être utilisée aussi bien en lecture qu'en écriture (initialisation ou modification). Le GFA Basic reconnaît la variable WINTAB qui contient l'adresse du premier élément de la table. On peut aussi manipuler cette table comme un tableau à deux dimensions, le premier indice désignant le numéro de la fenêtre (0 à 4) et le second la nature du paramètre à modifier (0 pour le handle, 1 pour l'attribut, 2 pour l'abscisse du coin supérieur gauche, 3 pour l'ordonnée du coin supérieur gauche, 4 pour la largeur, 5 pour la hauteur).

Un paramètre sur 12 bits significatifs que l'on nomme attribut de la fenêtre spécifie les éléments composant cette même fenêtre. Il est calculé, comme suit, chaque élément étant associé à un bit mis à un lorsque cet élément est présent :

- 1 = barre pour le nom de la fenêtre, cet attribut est toujours présent (bit 0 à 1);
- 2 = case de fermeture (bit 1) ;
- 4 = case plein écran (bit 2) ;
- 8 = barre de déplacement (bit 3) ;
- 16 = ligne d'information (bit 4) ;
- 32 = case de taille (bit 5) ;
- 64 = flèche haute de l'ascenseur vertical (bit 6) ;
- 128 = flèche basse de l'ascenseur vertical (bit 7) ;
- 256 = ascenseur vertical (bit 8) ;
- 512 = flèche gauche de l'ascenseur horizontal (bit 9) ;
- 1024 = flèche droite de l'ascenseur horizontal (bit 10) ;
- 2048 = ascenseur horizontal (bit 11).

WINTAB est composée comme suit :

- WINTAB+0 = handle de la première fenêtre ;
- WINTAB+2 = attributs de la première fenêtre ;
- WINTAB+4 = abscisse X de la première fenêtre ;
- WINTAB+6 = ordonnée Y de la première fenêtre ;
- WINTAB+8 = largeur de la première fenêtre ;
- WINTAB+10 = hauteur de la première fenêtre ;
- WINTAB+12
- à = identiques pour la deuxième fenêtre ;
- WINTAB+22

WINTAB+24

à = identiques pour la troisième fenêtre ;

WINTAB+34

WINTAB+36

à = identiques pour la quatrième fenêtre ;

WINTAB+46

WINTAB+48

à = inutilisé ;

WINTAB+50

WINTAB+52 = abscisse X de l'écran sans la barre de menu ;

WINTAB+54 = ordonnée Y de l'écran sans la barre de menu ;

WINTAB+56 = largeur de l'écran sans la barre de menu ;

WINTAB+58 = hauteur de l'écran sans la barre de menu ;

WINTAB+60 = abscisse X du point d'intersection des quatre fenêtres ;

WINTAB+62 = ordonnée Y du point d'intersection des quatre fenêtres ;

WINTAB+64 = abscisse X de l'origine pour les instructions graphiques ;

WINTAB+68 = ordonnée Y de l'origine pour les instructions graphiques.

W_HAND(#n)

n : expression numérique entière.

Cette fonction renvoie le handle, code GEM de la fenêtre n.

W_INDEX(#handle)

handle : expression numérique entière.

Cette fonction renvoie le numéro de la fenêtre associée à cet handle.

**EXEMPLE
DE PROGRAMMATION
DES FENETRES
ET DE LA SOURIS**

197

Chaque programme est donné dans son intégralité et décortiqué afin de vous expliquer comment fonctionne les diverses instructions. Les programmes fonctionnent dans la mesure du possible dans les trois résolutions de l'Atari ST.

VISITE GUIDÉE DES FENÊTRES

Ce programme gère simultanément tous les types d'événements qui peuvent se produire et affiche les résultats.

Programme principal

Les instructions suivantes sont facultatives; toutefois, elles faciliteront l'écriture du programme. En effet, lorsque les variables sont saisies (mode DEFLIST 2), elles prennent automatiquement le bon suffixe à l'affichage.

DEFBYT "numero,cx,cy,nb_coul"	désigne des octets ;
DEFBYT "fenetre"	désigne des octets ;
DEFWRD "a-z"	désigne des mots (16 bits).

Variables globales :

cx	facteur de résolution horizontal ;
cy	facteur de résolution vertical ;
nb_coul	nombre de couleurs ;
l&	calcul de la largeur des fenêtres ;
h&	calcul de la hauteur des fenêtres ;
men\$	variable de menu ;
c_a\$	code ASCII ;
c_c\$	code clavier.

Procédure init_menu

Cette procédure initialise le menu.

Variable locale :

i	indice de boucle.
---	-------------------

Procédure ouvre_fenetre

Cette procédure initialise les fenêtres.

Paramètres :

numero	numéro de la fenêtre ;
x&	abscisse de la fenêtre ;
y&	ordonnée de la fenêtre ;
l&	largeur de la fenêtre ;
h&	hauteur de la fenêtre.

Procédure resolution

Cette procédure initialise les facteurs de résolution.

Paramètres :

cx| facteur de résolution horizontal ;
cy| facteur de résolution vertical ;
nb_coul| nombre de couleurs.

Procédure souris

Cette procédure gère un événement souris.

Procédure touche

Cette procédure gère un événement clavier.

Procédure traitement

Cette procédure gère les événements du menu déroulant.

Procédure fenetre1

Cette procédure gère les événements relatifs à la fenêtre numéro 1.

Procédure fenetre2

Cette procédure gère les événements relatifs à la fenêtre numéro 2.

Procédure message

Cette procédure gère tous les événements simultanément.

Variables locales :

index| indice de l'événement ;
f\$ numéro de la fenêtre.

Procédure evenement

Cette procédure affiche la nature des événements dans la fenêtre numéro 3.

Paramètre :

ev\$ message de l'événement.

Procédure reduc_tableau

Cette procédure réduit un tableau à la taille réellement utilisée.

Paramètres :

dimension& dimension utile ;
tableau\$() tableau à réduire.

Variables locales :

i& indice de boucle ;
t_pro\$() tableau de stockage provisoire.

Programme :

```
' -----
'      MENU ET FENETRE
' -----
' DEFBYT "numero,cx,cy,nb_coul"
' DEFBYT "fenetre"
' DEFWRD "a-z"
'
resolution(cx|,cy|,nb_coul|)
l&=MUL(160,cx|)
h&=MUL(90,cy|)
ouvre_fenetre(1,0,19,l&,h&)
ouvre_fenetre(2,l&+1,19,l&,h&)
ouvre_fenetre(3,0,SUB(MUL(200,cy|),h&),MUL(l&,2),h&)
TOPW #2
PRINT AT(3,2);"bonjour"
CIRCLE 30,25,20
init_menu
MENU men$( )
ON MENU GOSUB traitement
ON MENU KEY GOSUB touche
ON MENU IBOX 1,0,19,l&,h& GOSUB fenetre1
ON MENU IBOX 2,l&+1,19,l&,h& GOSUB fenetre2
ON MENU BUTTON 1,1,1 GOSUB souris
ON MENU MESSAGE GOSUB message
DO
  ON MENU
LOOP
END
' -----
'      PROCEDURES ET FONCTIONS
' -----
PROCEDURE init_menu
  LOCAL i|
  DIM men$(50),etat|(50)
  i|=0
  REPEAT
    READ men$(i|)
    INC i|
  UNTIL men$(i|-1)="*"
  DEC i|
  men$(i|)=" "
  reduc_tableau(i|,men$( ))
  DATA Bureau, Informations
  DATA -----
```

```

DATA 1,2,3,4,5,6,""
DATA Fichier, Charger, Sauvegarder, Fermer, Quitter,""
DATA *
RETURN
'
PROCEDURE ouvre_fenetre(numero|,x&,y&,l&,h&)
OPENW #numero|,x&,y&,l&,h&,&X111111111111
TITLEW #numero|,"fenetre "+CHR$(48+numero|)
RETURN
'
PROCEDURE resolution(VAR cx|,cy|,nb_coul|)
SELECT (XBIOS(4))
CASE 0
cx|=1
cy|=1
nb_coul|=16
CASE 1
cx|=2
cy|=1
nb_coul|=4
CASE 2
cx|=2
cy|=2
nb_coul|=2
ENDSELECT
RETURN
'
PROCEDURE souris
evenement("souris")
RETURN
'
PROCEDURE touche
c_a$=STR$(BYTE(MENU(14)))
c_c$=STR$(BYTE(ROR&(MENU(14),8)))
evenement("touche. code ASCII : "+c_a$+" code_clavier :
"+c_c$)
RETURN
'
PROCEDURE traitement
message
evenement("option du menu : "+men$(MENU(0)))
IF men$(MENU(0))=" Quitter" THEN
CLOSE #1
CLOSE #2
CLOSE #3
MENU KILL
STOP

```

```

    ouvre_fenetre(1,0,19,1&,h&)
    ouvre_fenetre(2,1&+1,19,1&,h&)
    ouvre_fenetre(3,0,SUB(MUL(200,cy|),h&),MUL(1&,2),h&)
    MENU men$()
ELSE
    etat|(MENU(0))=SUB(1,etat|(MENU(0)))
    MENU MENU(0),etat|(MENU(0))
ENDIF
MENU OFF
RETURN
,
PROCEDURE fenetre1
    IF fenetre|<>1 THEN
        fenetre|=1
        evenement("souris dans la fenetre 1")
    ENDIF
RETURN
,
PROCEDURE fenetre2
    IF fenetre|<>2 THEN
        fenetre|=2
        evenement("souris dans la fenetre 2")
    ENDIF
RETURN
,
PROCEDURE message
    LOCAL index|,f$
    index|=W_INDEX(#MENU(4))
    f$=CHR$(48+index|)
    SELECT MENU(1)
CASE 10
    evenement("menu déroulant")
CASE 20
    evenement("rafraichissement de l'écran")
CASE 21
    window|=index|
    evenement("fenetre "+f$+" active")
CASE 22
    evenement("fermeture de la fenetre "+f$)
CASE 23
    evenement("taille maximale pour la fenetre "+f$)
CASE 24
    evenement("bouton poussoir de la fenetre "+f$)
CASE 25
    evenement("déplacement du bouton poussoir horizontal de
la fenetre "+f$)
CASE 26

```

```

        evenement("déplacement du bouton poussoir vertical de la
fenetre "+f$)
    CASE 27
        evenement("modification de la taille de la fenetre "+f$)
    CASE 28
        evenement("déplacement de la fenetre "+f$)
    ENDSELECT
RETURN
'
PROCEDURE evenement(ev$)
    TOPW #3
    PRINT ev$
    TOPW #window|
RETURN
'
PROCEDURE reduc_tableau(dimension&,VAR tableau$())
    LOCAL i&
    DIM t_pro$(dimension&)
    FOR i&=0 TO dimension&
        t_pro$(i&)=tableau$(i&)
    NEXT i&
    SWAP t_pro$(),tableau$()
    ERASE t_pro$()
RETURN

```

FICHIERS ET REPERTOIRES

205

FICHIERS

BGET	INPUT\$	PUT
BLOAD	KILL	RECALL
BPUT	LINE INPUT	RECORD
BSAVE	LOC	RELSEEK
CLOSE	LOF	RSET
EOF	LSET	SEEK
EXIST	NAME	STORE
FIELD	OPEN	TOUCH
GET	OUT	WRITE
INP	PRINT	
INPUT	PRINT USING	

BGET [#]n,adr,nb

n,nb : expressions numériques entières ;

adr : mot long.

Cette instruction charge nb octets à l'adresse adr en les lisant sur le canal n.

Voir également les instructions **BPUT**, **BLOAD** et **BSAVE**.

BLOAD fichier[,adr]

fichier : expression alphanumérique ;

adr : mot long.

Cette instruction charge le fichier à l'adresse adr. Si celle-ci n'est pas spécifiée, alors l'adresse indiquée lors de la sauvegarde est utilisée.

Voir également les instructions **BSAVE**, **BPUT** et **BGET**.

BPUT [#]n,adr,nb

n,nb : expressions numériques entières ;

adr : mot long.

Cette instruction envoie sur le canal n nb octets de la zone mémoire débutant à l'adresse adr.

Voir également les instructions **BGET**, **BSAVE** et **BLOAD**.

BSAVE fichier,adr,n

fichier : expression alphanumérique ;

adr : mot long ;

n : expression numérique entière.

Cette instruction sauve, dans le fichier indiqué, une zone mémoire de n octets à partir de l'adresse adr.

Voir également les instructions **BLOAD**, **BPUT** et **BGET**.

CLOSE [[#]n]

n : expression numérique entière.

Cette instruction referme le canal n, précédemment ouvert avec **OPEN**. Si le paramètre n n'est pas spécifié, tous les canaux ouverts sont refermés.

EOF([#]n)

n : expression numérique entière.

Cette fonction renvoie TRUE (-1) si l'on se trouve à la fin du fichier ouvert sous le canal n et retourne FALSE (0) dans les autres cas.

EXIST(fichier)

fichier : expression alphanumérique.

Cette instruction teste l'existence d'un fichier (ou d'une catégorie de fichiers). Elle retourne TRUE (-1) si le fichier (ou les fichiers) existe, ou bien FALSE (0) si le fichier (ou les fichiers) n'existe pas. Toutes les spécifications suivantes sont permises simultanément :

- le nom de l'unité de disquettes (ou de disque dur) ;
- le nom des fichiers.
 - un ? remplace une lettre quelconque dans le nom des fichiers, par exemple, EXIST("?ILI.LIB") teste l'existence de tous les fichiers dont le nom commence par une lettre quelconque et s'achève par ILI.LIB (LILI.LIB et TILI.LIB seraient pris en compte) ;
 - une * remplace une partie quelconque dans le nom des fichiers, par exemple, EXIST("ZO*.*") teste l'existence de tous les fichiers dont le nom commence par ZO.

FIELD [#]n,longueur1 AS variable1[,longueur2 AS variable2,...]**FIELD [#]n,longueur1 AT(adr1),[longueur2 AT(adr2)...]**

n,longueur1,longueur2... : expressions numériques entières ;

variable1,variable2... : variables alphanumériques ;

adr1,adr2... : mots longs.

Cette instruction possède deux modalités mixables dans une même instruction :
 - la modalité AS définit les champs de données d'un enregistrement du fichier ouvert sous le numéro de canal n. Une seule définition est permise. La somme des longueurs ne doit en aucun cas dépasser la longueur totale d'un enregistrement (spécifiée lors de l'ouverture du fichier). En clair, prenons un exemple précis :

FIELD #5,20 AS fichier,62 AS adresse\$,8 AS telephone\$

divise la fiche en trois champs : le nom, l'adresse et le numéro de téléphone. Les variables fichier, adresse\$ et telephone\$ doivent être affectées avec LSET ou RSET.

- la modalité AT est presque identique à FIELD AS. Les adresses (adr1, adr2...) sont des pointeurs sur des variables dont la longueur correspond à celle indiquée (longueur1, longueur2...). Ainsi, les variables numériques peuvent être directement intégrées dans un enregistrement.

GET [#]n[,i]

n,i : expressions numériques entières.

Cette instruction lit un enregistrement sur le fichier à accès direct ouvert sous le numéro de canal n. Le paramètre optionnel i correspond au numéro de l'enregistrement dans le fichier (compris entre 1 et le nombre d'enregistrements). Si i est manquant alors, l'enregistrement suivant, pointé par LOC(#n)+1, est lu.

INP(#n)

n : expression numérique entière.

Cette fonction lit un octet du fichier ouvert comme canal n.

KILL fichier

fichier : expression alphanumérique.

Cette instruction détruit le fichier considéré. Toutes les spécifications suivantes sont permises simultanément :

- le nom de l'unité de disquettes (ou de disque dur), par exemple, KILL A: détruit le premier des fichiers de l'unité A ;

- le nom des fichiers :

- un ? remplace une lettre quelconque dans le nom des fichiers, par exemple, KILL ?ILI.LIB détruit le premier des fichiers dont le nom commence par une lettre quelconque et s'achève par ILI.LIB (si LILI.LIB et TILI.LIB sont présents, seul LILI.LIB serait détruit) ;

- une * remplace une partie quelconque dans le nom des fichiers, par exemple, KILL ZO*. * détruit le premier des fichiers dont le nom commence par ZO ;

- KILL utilise également le système de dossiers, par exemple, KILL "dossier1\dossier2\dossier3*.GFA" détruit le premier fichier ayant pour suffixe .GFA du dossier3, contenu dans le dossier2, lui-même contenu dans le dossier1 du dossier principal (indiqué par \, qui doit précéder tout nom de dossier).

Il est important de noter que seul le premier fichier répondant aux spécifications indiquées est détruit.

LOC([#]n)

n : expression numérique entière.

Cette fonction renvoie la valeur du pointeur du fichier ouvert comme canal n.

LOF([#]n)

n : expression numérique entière.

Cette fonction retourne la taille occupée par le fichier ouvert comme canal n.

LSET champ=chaîne

champ : variable alphanumérique ;

chaîne : expression alphanumérique.

Cette instruction affecte la chaîne à la variable champ en la justifiant à gauche. Si chaîne a une longueur inférieure à celle de champ, la fin est complétée de blancs. Si chaîne a une longueur supérieure à celle de champ, chaîne est tronquée à droite. La variable champ doit auparavant avoir été définie, et elle gardera sa longueur initiale. LSET sert surtout avec FIELD, les données numériques doivent être converties avec MKI\$, MKL\$, MKS\$, MKF\$ et MKD\$ avant leur affectation grâce à LSET.

NAME fichier0 AS fichier1

fichier0, fichier1 : expressions alphanumériques.

Cette instruction renomme fichier0 en lui donnant le nom fichier1. Les spécifications suivantes des noms de fichiers sont permises simultanément :

- le nom de l'unité de disquettes (ou de disque dur) qui doit être identique pour les deux noms de fichiers ;

- NAME utilise le système de dossiers :

Par exemple : NAME "A\dossier1\LILI" AS "dossier2\TILI.DOC" transfère le fichier "LILI" contenu dans le dossier1 du dossier principal de l'unité A dans le dossier2 du dossier principal de l'unité A en le renommant "TILI.DOC". L'instruction RENAME est identique à NAME.

OPEN mode,[#]n,canal[,longueur]

mode, canal : expressions alphanumériques ;

n, longueur : expressions numériques entières.

Cette instruction ouvre un canal sous le numéro n (entre 0 et 99) :

- le canal peut être un fichier accessible avec toutes les spécifications possibles des noms de fichiers. OPEN utilise également le système de dossiers. Par exemple, OPEN "C:\dossier1\dossier2\dossier3\DATA.DAT" ouvre le fichier "DATA.DAT" de l'unité C du dossier3, contenu dans le dossier2, lui-même contenu dans le dossier1 du dossier principal (indiqué par \, qui doit précéder tout nom de dossier). Les modes d'accès sont les suivants :

- O crée un fichier en écriture ;

- I ouvre un fichier en lecture ;

- A autorise l'ajout de données à un fichier ;

- U ouvre un fichier déjà existant en lecture et en écriture ;

- R ouvre un fichier en accès direct. Dans ce cas, la longueur d'un enregistrement peut être précisée, elle sera de 128 par défaut ;

- le canal peut être également :

- CON: pour la console ;

- LST: ou PRN: pour l'imprimante ;

- AUX: pour l'interface série ;

- MID: pour la sortie MIDI ;

- VID: pour la console en mode "transparent" (les caractères de contrôles ne sont pas exécutés) ;

- IKBD: pour l'accès direct au processeur du clavier 6301. Attention dangereux !

Dans ces cas, le mode d'accès est ignoré et l'on peut indiquer une chaîne vide.

Par exemple, OPEN "R",#5,"agenda",90 ouvre le fichier "agenda" en accès direct et donne à chaque enregistrement ou fiche une longueur de 90 octets.

OUT #n,n1[,n2..]

n : expression numérique entière ;

n1,n2... : octets.

Cette fonction écrit un ou plusieurs octets dans le fichier ouvert comme canal n.

PUT [#]n[,i]

n,i : expressions numériques entières.

Cette instruction écrit un enregistrement sur le fichier à accès direct ouvert sous le numéro de canal n. Le paramètre optionnel i correspond au numéro de l'enregistrement dans le fichier (compris entre 1 et le nombre d'enregistrements plus un). Si i est manquant, alors l'enregistrement est placé en fin de fichier.

RECALL #n,tab,nb1,nb2

n,nb1 : expressions numériques entières ;

nb2 : variable mot long ;

tab : tableau alphanumérique.

Cette instruction initialise un tableau alphanumérique avec les nb1 premiers éléments du fichier ouvert comme canal n (nb=-1 provoque le chargement du tableau tout entier). Si le tableau ou le fichier sont trop petits, le chargement s'effectue sans erreur : le plus petit impose le nombre de chaînes de caractères transférées, ce dernier est renvoyé dans la variable nb2.

RECORD #n,i

n,i : expressions numériques entières.

Cette instruction fixe le numéro du prochain enregistrement à traiter.

RELSEEK [#]n,d

n,d : expressions numériques entières.

Cette instruction déplace le pointeur du fichier ouvert comme canal n de la valeur du déplacement d en octets spécifié (le déplacement peut être négatif). Attention, on doit rester dans les limites du fichier!

RSET champ=chaîne

champ : variable alphanumérique ;

chaîne : expression alphanumérique.

Cette instruction affecte la chaîne à la variable champ en la justifiant à droite. Si chaîne a une longueur inférieure à celle de champ, le début est complété de blancs. Si chaîne a une longueur supérieure à celle de champ, chaîne est tronquée à gauche. La variable champ doit auparavant avoir été définie et elle gardera sa longueur initiale. RSET sert surtout avec FIELD, les données numériques doivent être converties avec MKI\$, MKL\$, MKS\$, MKF\$ et MKD\$ avant leur affectation grâce à RSET.

SEEK [#]n,i

n,i : expressions numériques entières.

Cette instruction place le pointeur du fichier ouvert comme canal n sur le ième octet du fichier. Si i est négatif, le pointeur est placé sur le i^e octets du fichier, mais en partant de la fin. Attention, on doit rester dans les limites du fichier!

STORE #n,tab,nb

n,nb : expressions numériques entières ;

tab : tableau alphanumérique.

Cette instruction sauvegarde les nb premiers éléments d'un tableau alphanumérique dans le fichier ouvert comme canal n. Les éléments sont séparés par CR/LF.

TOUCH[#]n

n : expression numérique entière.

Cette instruction actualise l'heure et la date du fichier ouvert comme canal n.

WRITE #n,expression1[(ou ;)expression2...]

expression1,expression2... : expressions ;

n : expression numérique entière.

Cette instruction sauvegarde sur le fichier séquentiel ouvert sous le numéro de canal n les données spécifiées. Les données ainsi sauvegardées seront relues correctement par l'instruction INPUT #n,liste_de_variables.

PRINT #n[,expression1[(ou ; ou ')expression2...]

expression1,expression2... : expressions ;

n : expression numérique entière.

Cette instruction écrit des données sur le fichier séquentiel ouvert sous le numéro de canal. Il faut cependant remarquer que toute la ligne de données transmise par l'instruction PRINT #n sera relue d'un seul tenant par l'instruction INPUT #n.

PRINT #n,USING format,expression1[(ou ;)expression2...]

format : expression alphanumérique ;

expression1,expression2... : expressions ;

n : expression numérique entière.

Cette instruction transmet des données dans un format donné au fichier séquentiel ouvert sous le numéro de canal n. La ligne de données transmise par l'instruction PRINT #n,USING sera relue d'un seul tenant par l'instruction INPUT #n.

- # remplace un chiffre ;
- . détermine la position du point décimal ;
- + oblige l'affichage des signes "+" ;
- réserve de la place pour un signe "—" éventuel ;
- * tous les 0 en tête de chaîne sont remplacés par des * ;
- \$\$ affiche un \$ en tête de chaîne ;
- , affiche la virgule des milliers ;
- ~~~~ format E+nn ;
- ~~~~~ format E+nnn ;
- ! seul le premier caractère est affiché ;
- & l'intégralité de la chaîne est affichée ;
- \. \ affiche une chaîne dont la longueur est équivalente aux nombre de points entre les deux séparateurs plus ces deux derniers ;
- _ affiche le caractère suivant.

INPUT #n,var1[,var2...]

var1,var2... : variables ;

n : expression numérique entière.

Cette instruction permet de lire des données sur le fichier séquentiel ouvert sous le numéro de canal n et de les placer dans les variables indiquées (var1,var2...).

LINE INPUT #n,var1[,var2...]

var1,var2... : variables alphanumériques ;

n : expression numérique entière.

Cette instruction permet de lire sur le fichier séquentiel ouvert sous le numéro de canal n une ligne entière jusqu'au retour chariot qui est le seul caractère séparateur reconnu, autorisant ainsi les ponctuations telles que la virgule ou le point-virgule qui ne sont pas acceptés par l'instruction INPUT.

INPUT\$(m,#n)

m,n : expressions numériques entières.

Cette instruction retourne m caractères lus à partir du fichier séquentiel ouvert sous le numéro de canal n.

REPERTOIRES

CHDIR
CHDRIVE
DFREE
DIR
DIR\$

FGETDTA
FILES
FILESELECT
FSETDTA
FSFIRST

FSNEXT
MKDIR
RMDIR

CHDIR dossier

dossier : expression alphanumérique.

Cette instruction détermine le dossier sous lequel on désire se placer. Le chemin permettant d'y accéder doit être indiqué de la manière suivante : CHDIR "dossier1\dossier2\dossier3" on se place ainsi dans le dossier3, contenu dans le dossier2, lui-même contenu dans le dossier1 du dossier principal (indiqué par \ qui doit précéder tout nom de dossier).

CHDRIVE n

n : expression numérique entière.

Cette instruction spécifie l'unité de disquette (ou de disque dur) qui sera prise par défaut comme unité courante. Le paramètre n varie entre 0 et 15 (A à P).

DFREE(n)

n : expression numérique entière.

Cette fonction renvoie la place mémoire disponible sur l'unité n de disquettes (ou de disque dur). Le paramètre n varie entre 0 et 15 (A à P).

DIR [fichier[TO destination]]

fichier,destination : expressions alphanumériques.

Cette instruction liste les fichiers spécifiés. Toutes les spécifications suivantes sont permises simultanément :

- le nom de l'unité de disquettes (ou de disque dur), par exemple, DIR A: liste l'unité A ;

- le nom des fichiers :

- un ? remplace une lettre quelconque dans le nom des fichiers, par exemple, DIR ?ILI.LIB liste tous les fichiers dont le nom commence par une lettre quelconque et s'achève par ILI.LIB (LILI.LIB et TILI.LIB seraient pris en compte) ;

- une * remplace une partie quelconque dans le nom des fichiers, par exemple, DIR ZO*.* liste tous les fichiers dont le nom commence par ZO quelle que soit l'extension du fichier ; DIR utilise également le système de dossiers, par exemple, DIR "\dossier1\dossier2\dossier3" liste le dossier3, contenu dans le dossier2, lui-même contenu dans le dossier1 du dossier principal (indiqué par \, qui doit précéder tout nom de dossier).

- L'option TO destination permet d'indiquer un canal de sortie autre que l'écran, LST: pour l'imprimante, ou un nom de fichier pour la sauvegarde sur disque.

DIR\$(n)

n : expression numérique entière.

Cette fonction renvoie le dossier actif de l'unité n. DIR\$ retourne 0 si l'on se trouve sous le dossier principal. Le paramètre n varie entre 0 et 15 (A à P).

FGETDTA()

Cette fonction renvoie l'adresse de la DTA.

FILES [fichier[TO destination]]

fichier,destination : expressions alphanumériques.

Cette instruction liste les fichiers spécifiés tout comme DIR mais la taille, l'heure et la date de création sont indiquées en supplément. Toutes les spécifications suivantes sont permises simultanément :

- le nom de l'unité de disquettes (ou de disque dur), par exemple, FILES A liste l'unité A ;

- le nom des fichiers :

- un ? remplace une lettre quelconque dans le nom des fichiers, par exemple, FILES ?ILI.LIB liste tous les fichiers dont le nom commence par une lettre quelconque et s'achève par ILI.LIB (LILI.LIB et TILI.LIB seraient pris en compte) ;

- une * remplace une partie quelconque dans le nom des fichiers, par exemple, FILES ZO*.* liste tous les fichiers dont le nom commence par ZO ;

- FILES utilise également le système de dossiers, par exemple, FILES "\dossier1\dossier2\dossier3" liste le dossier3, contenu dans le dossier2, lui-même contenu dans le dossier1 du dossier principal (indiqué par \, qui doit précéder tout nom de dossier).

- L'option, TO destination, permet d'indiquer un canal de sortie autre que l'écran, LST: pour l'imprimante, ou un nom de fichier pour la sauvegarde sur disque.

FILESELECT chemin,fichier,selection

chemin,fichier : expressions alphanumériques ;

selection : variable alphanumérique.

Cette instruction permet la sélection de fichiers grâce à une boîte de dialogue. Le chemin suivi est indiqué par "fichiers" (la spécification minimale "*.*)" sélectionne tous les fichiers du dossier principal). Le second fichier (fichier_defaut) indique celui pris par défaut et la variable selection\$ contient au retour le fichier choisi.

Par exemple : FILESELECT "\GFA*.GFA","DEMO.GFA",selection proposera de choisir parmi tous les fichiers du dossier GFA dont le suffixe est .GFA (programmes GFA standards en version 2.02) celui qui sera placé dans la variable selection, et choisira DEMO.GFA par défaut.

FSETDTA(adr)

adr : mot long.

Cette fonction fixe la DTA à l'adresse adr. La DTA présente la structure suivante :

octets	contenu
1 à 21	réservé ;
22	attributs ;
23 à 24	heure ;
25 à 26	date ;
27 à 30	longueur ;
31 à 44	nom du fichier.

Les attributs sont codés comme suit (le bit à 1 active la propriété) :

bit	signification
0	fichier protégé en écriture ;
1	fichier caché ;
2	fichier système ;
3	unité disque ;
4	dossier ;
5	archivage.

FSFIRST(critere,attribut)

critere : expression alphanumérique ;

attribut : expression numérique entière.

Cette fonction recherche le premier fichier correspondant au critère indiqué.

Tous les critères suivants sont permis simultanément :

- le nom de l'unité de disquettes (ou de disque dur) ;
- le nom des fichiers :
 - un ? remplace une lettre quelconque dans le nom des fichier ;
 - une * remplace une partie quelconque dans le nom des fichiers ;
- le système de dossiers.

Cette fonction renvoie une valeur booléenne qui indique si la recherche a réussi (TRUE) et initialise la variable attribut.

FSNEXT()

Cette fonction recherche le prochain fichier qui correspond au critère spécifié dans FSFIRST. Elle renvoie une valeur booléenne qui indique si la recherche a réussi (TRUE) et initialise la variable attribut.

MKDIR dossier

dossier : expression alphanumérique.

Cette instruction crée un nouveau dossier, par exemple, MKDIR "B:\dossier1\dossier2\dossier3" crée sur le disque B, le dossier3, contenu dans le dossier2 lui-même contenu dans le dossier1 du dossier principal (indiqué par \, qui doit précéder tout nom de dossier).

RMDIR dossier

dossier : expression alphanumérique.

Cette instruction détruit un dossier par exemple, MKDIR "C:\dossier1\dossier2\dossier3" détruit sur le disque C le dossier3 contenu dans le dossier2 lui-même contenu dans le dossier1 du dossier principal (indiqué par \, qui doit précéder tout nom de dossier). Le dossier n'est détruit que s'il est vide de fichiers ou de sous-dossiers.

BIOS, XBIOS ET GEMDOS

217

LE BIOS

Le BIOS gère les fonctions primaires d'entrées/sorties du système. Concrètement, il regroupe l'édition à l'écran et sur imprimante, les communications (export et import) avec l'interface série RS-232C et l'unité de disquette (ou disque dur). Le code rendu de l'opération est contenu dans le registre D0. L'appel s'effectue de la manière suivante :

~BIOS(code,par1,par2...)

ou bien alors

var=BIOS(code,par1,par2...)

var : variable numérique entière ;

code : expression numérique entière ;

par1,par2... : mots longs ou mots.

Cette fonction appelle la routine BIOS désignée par le code opération, en lui passant une liste de paramètres qui sont, soit des mots précédés par W: (pris par défaut), soit des mots longs précédés par L:. Toute erreur dans le passage des paramètres, portant sur leur valeur ou leur type (L: ou W:), peut entraîner un crash général de la machine. Au retour, la variable var contient la valeur de retour ou le code d'erreur, précédemment stockée dans D0.

Voici, maintenant, une liste exhaustive des fonctions BIOS :

FONCTION 0 (\$00) : GETMPB

Comme son nom l'indique, GETMPB va chercher le MPB (Memory Parameter Bloc, bloc mémoire de paramètre) et le place dans un bloc de 12 octets. Pour des raisons que vous comprendrez aisément, cette fonction est à prendre avec précaution si l'on tient à ne pas mélanger les pointeurs qui servent à la gestion mémoire du GEMDOS.

Voici comment sont organisés ces 12 octets :

Octets 0 à 3 : MFL, c'est-à-dire la mémoire libre, Memory Free List ;

Octets 4 à 7 : MAL, c'est-à-dire la mémoire Memory Allocated List ;

Octet 8 à B : RP, c'est-à-dire le pointeur du descripteur de la mémoire, Roving Pointer.

Le descripteur de mémoire, MD, est composé de 16 octets se divisant en 4 adresses. Voici la structure du descripteur de la mémoire :

Octets 0 à 3 : LINK, adresse du descripteur suivant ;

Octets 4 à 7 : START, adresse de la mémoire libre ;

Octets 8 à B : LENGHT, longueur de la mémoire libre ;

Octets C à F : OWN, adresse de la page de base du processus appelant.

En GFA Basic :

BIOS(0,L:adr)

adr : mot long.

L'adresse des 12 octets du MPB est pointée par adr.

FONCTION 1 (\$01) : BCONSTAT

Cette fonction détermine l'état d'un périphérique d'entrée. Il existe deux états : actif et inactif. Cette fonction renvoie donc -1 ou 0 selon qu'il y ait un caractère en entrée ou non. Le périphérique est codé dans la variable dev de la manière suivante :

- 0 pour PRT: l'imprimante ;
- 1 pour AUX: l'interface RS-232C ;
- 2 pour CON: la console (écran + clavier) ;
- 3 pour MID: l'interface MIDI ;
- 4 pour IKBD: le processeur clavier 6301 ;
- 5 pour la sortie écran.

Attention, programmer le processeur clavier 6301 peut s'avérer dangereux

En GFA Basic :

BIOS(1,W:dev)

dev : mot.

dev est le périphérique d'entrée.

FONCTION 2 (\$02) : BCONIN

Cette fonction va chercher un caractère sur le périphérique d'entrée. Elle renvoie le code du caractère. Le périphérique est codé dans la variable dev de la manière suivante :

- 0 pour PRT: l'imprimante ;
- 1 pour AUX: l'interface RS-232C ;
- 2 pour CON: la console (écran + clavier) ;
- 3 pour MID: l'interface MIDI ;
- 4 pour IKBD: le processeur clavier 6301 ;
- 5 pour la sortie écran.

Attention, programmer le processeur clavier 6301 peut s'avérer dangereux

En GFA Basic :

BIOS(2,W:dev)

dev : mot.

dev est le périphérique d'entrée.

FONCTION 3 (\$03) : BCONOUT

Cette fonction envoie un caractère sur le périphérique spécifié par le paramètre dev. Ce dernier est codé de la manière suivante :

- 0 pour PRT: l'imprimante ;
- 1 pour AUX: l'interface RS-232C ;
- 2 pour CON: la console (écran + clavier) ;
- 3 pour MID: l'interface MIDI ;
- 4 pour IKBD: le processeur clavier 6301 ;
- 5 pour la sortie écran.

Attention, programmer le processeur clavier 6301 peut s'avérer dangereux

En GFA Basic :

BIOS(3,W:dev,W:code)

dev,code : mots.

Le code ASCII du caractère est contenu dans code, et dev est le périphérique.

FONCTION 4 (\$04) : RWABS

Cette fonction procède à l'écriture ou la lecture d'un secteur sur le disque. Elle renvoie un code d'erreur si cette manipulation est impossible ou se passe mal.

La fonction nécessite cinq paramètres :

la modalité d'utilisation, est représentée, ici par un drapeau flag (sur 16 bits):

flag = 0 pour une lecture simple ;

flag = 1 pour une écriture simple ;

flag = 2 pour une lecture sans regarder si le disque a été changé ;

flag = 3 pour une écriture sans regarder si le disque a été changé ;

l'adresse adr du tableau où l'on stockera et/ou lira les informations du disque ;

le nombre de secteurs à lire ou écrire, n1 ;

le numéro du secteur d'où débiteront les opérations de lectures ou d'écriture, n2 ;

le périphérique, dev :

dev = 0 pour le lecteur de disquettes A ;

dev = 1 pour le lecteur de disquettes B ;

dev = 2 pour le disque dur ou une de ses partitions.

En GFA Basic :

BIOS(4,W:flag,L:adr,W:n1,W:n2,W:dev)

flag,n1,n2,dev : mots ;

adr : mot long.

FONCTION 5 (\$05) : SETEXEC

Cette fonction lit ou modifie un vecteur d'exception du 68000. Ce dernier possède 256 vecteurs d'exception; 8 sont utilisés et codés sous Gem en 8 vecteurs numérotés ainsi :

vecteur no \$100 à l'adresse \$400 ;

vecteur no \$101 à l'adresse \$404 ;

vecteur no \$102 à l'adresse \$408 ;

vecteur no \$103 à l'adresse \$40C ;

vecteur no \$104 à l'adresse \$410 ;

vecteur no \$105 à l'adresse \$414 ;

vecteur no \$106 à l'adresse \$418 ;

vecteur no \$107 à l'adresse \$41C.

On passe deux paramètres à cette fonction, le numéro du vecteur et l'adresse de la routine qui va se substituer au vecteur d'origine. Si cette adresse vaut -1, la fonction se contente de faire une lecture simple du vecteur d'exception spécifié et de le renvoyer.

En GFA Basic :

BIOS(5,W:n,L:adr)

n : mot ;

adr : mot long

Le numéro du vecteur d'exception est n et adr représente l'adresse.

FONCTION 6 (\$06) : TICKCAL

Cette fonction détermine le temps écoulé entre deux impulsions de l'horloge du système. Ce temps est exprimé en millisecondes. Le ST renvoie systématiquement, 20 ms ce qui correspond à une fréquence de 50 Hz.

En GFA Basic :

BIOS(6)

FONCTION 7 (\$07) : GETBPB

La fonction retourne l'adresse mémoire du descripteur de disque. Les informations du BPB sont codées sur 16 bits comme suit :

taille des secteurs (512) ;

nombre de secteurs par cluster (2) ;

nombre de clusters (1024) ;

taille du répertoire (7) ;

taille de la FAT (5) ;

numéro du secteur de la seconde FAT (6) ;

numéro du secteur du premier cluster (18) ;

nombre de clusters de données sur la disquette (351 en SF et 711 en DF) ;
drapeaux.

Les indications entre parenthèses correspondent aux disquettes simple et double face. Le périphérique est codé de la manière suivante :

dev = 0 pour le lecteur de disquettes A ;

dev = 1 pour le lecteur de disquettes B ;

dev = 2 pour le disque dur ou une de ses partitions.

En GFA Basic :

BIOS(7,W:dev)

dev : mot.

dev est le périphérique.

FONCTION 8 (\$08) : BCOSTAT

La fonction regarde si le périphérique est prêt à recevoir des données. Il existe deux états: actif et inactif. Cette fonction renvoie donc -1 ou 0 selon qu'il y a un caractère en entrée ou non. Le périphérique est codé dans la variable dev de la manière suivante :

0 pour PRT: l'imprimante ;

1 pour AUX: l'interface RS-232C ;

2 pour CON: la console (écran + clavier) ;

3 pour MID: l'interface MIDI ;

4 pour IKBD: le processeur clavier 6301 ;

5 pour la sortie écran.

En GFA Basic :

BIOS(8,W:dev)

dev : mot long.

dev est le périphérique.

FONCTION 9 (\$09) : MEDIACH

Cette fonction détermine si une disquette a été enlevée du lecteur de disquettes. Il est bien évident que si le périphérique est un disque dur, la fonction répondra qu'il est impossible d'enlever une disquette.

0 = il s'agit d'une unité fixe ;

1 = la disquette a peut-être été changée ;

2 = la disquette a été changée.

Le périphérique prendra l'une des valeurs suivantes :

dev = 0 pour le lecteur de disquettes A ;

dev = 1 pour le lecteur de disquettes B ;

dev = 2 pour le disque dur ou une de ses partitions.

En GFA Basic :

BIOS(9,W:dev)

dev : mot long.

dev est le périphérique.

FONCTION 10 (\$0A) : DRVMAP

Cette fonction détermine les unités de masse qui sont reliées au ST. l'unité A est représentée par le bit 0, l'unité B par le bit 1... Lorsque le bit est à 1, cela signifie que l'unité logique est connectée.

En GFA Basic :

BIOS(10)

FONCTION 11 (\$0B) : KBSHIFT

Cette fonction détermine ou modifie l'état des touches spéciales : SHIFT, CONTROL, ALTERNATE, ALT+CLR/HOME (bouton droit de la souris), ALT+INS (bouton gauche de la souris) et CAPS LOCK. On transmet à la fonction la modalité d'utilisation. Si ce dernier est -1, la fonction KBSHIFT permet de lire l'état d'une touche. En revanche, si la modalité est un nombre compris entre 0 et 255, la fonction fait comme si l'on appuyait sur les touches correspondant à la combinaison des bits.

Le mode est un mot de 16 bits codé de la manière suivante :

Bit 0 &X.....1 touche SHIFT droite

Bit 1 &X.....1. touche SHIFT gauche

Bit 2 &X.....1.. touche CONTROL

Bit 3 &X....1... touche ALTERNATE

Bit 4 &X...1.... touche CAPS LOCK

Bit 5 &X..1..... bouton droit de la souris (ALT + CLR/HOME)

Bit 6 &X.1..... bouton gauche de la souris (ALT + INS)

Bit 7 &X1..... inutilisé

Lorsqu'un bit est à 1, cela signifie que la touche est enfoncée.

En GFA Basic :

BIOS(11,W:mode)

mode : mot.

mode est le mot de 16 bits, ou -1 pour une lecture simple.

LES FONCTIONS DU BIOS ETENDU : XBIOS

Le XBIOS désigne l'extension du BIOS. Ce sont des fonctions complétant le système d'exploitation afin d'avoir accès aux possibilités spéciales de l'Atari ST. Les fonctions sont au nombre de 40 (de 0 à 27 en hexadécimal). L'appel se fait suivant les modalités :

~XBIOS(code,par1,par2...)

ou bien alors

var=XBIOS(code,par1,par2...)

var : variable numérique entière ;

code : expression numérique entière ;

par1,par2... : mots longs ou mots.

Cette fonction appelle la routine XBIOS désignée par le code opération, en lui passant une liste de paramètres qui sont soit des mots précédés par W: (pris par défaut), soit des mots longs précédés par L:. Toute erreur dans le passage des paramètres, portant sur leur valeur ou leur type (L: ou W:), peut entraîner un crash général de la machine. Au retour, la variable var contient la valeur de retour ou le code d'erreur, précédemment stockée dans D0.

FONCTION 0 (\$00) : INITMOUS

Cette fonction initialise le gestionnaire de la souris. En fait, elle permet essentiellement de détourner le pointeur pour qu'il pointe sur des routines de gestion de la souris écrites par l'utilisateur. La souris est gérée par le processeur clavier ; le vecteur est appelé par le Mouse Report du clavier. Il faut envoyer comme paramètre le mode d'utilisation de la souris, puis deux mots longs contenant respectivement l'adresse de la routine qui sera appelée par le Mouse Report et un pointeur vers le bloc de paramètres.

Mode d'utilisation :

mode = 0 --> souris déconnectée ;

mode = 1 --> souris connectée en mode relatif ;

mode = 2 --> souris connectée en mode absolu ;

mode = 3 --> inutilisé ;

mode = 4 --> souris connectée en mode clavier.

Pointeur du bloc de paramètres :

octet 1 : souris en 0,0 en haut ou en bas (0 ou 1) ;

octet 2 : boutons de la souris actifs ;

octet 3 : coordonnées en x ;

octet 4 : coordonnées en y.

D'autres octets d'informations peuvent compléter ce bloc.

En GFA Basic :

XBIOS(0,W:mode,L:adr1,L:adr2)

mode : mot ;

adr1,adr2 : mots longs.

mode est le mode d'utilisation, adr1 est l'adresse du bloc de paramètres et adr2, l'adresse de la routine.

Attention, cette routine déconnecte la gestion de la souris par GEM.

FONCTION 1 (\$01) : SSBK

Cette fonction réserve de la place dans le haut de la mémoire. Il faut déclarer le nombre d'octets à réserver. Cette fonction est appelée juste avant l'initialisation du système d'exploitation, lors du BOOT_ROOM. Elle sert le plus souvent pour l'utilisation d'une cartouche de mémoire morte (ROM).

En GFA Basic :
inutile.

FONCTION 2 (\$02) : PHYSBAS

Cette fonction renvoie l'adresse physique de la mémoire écran, c'est-à-dire le début de la RAM vidéo gérée par le Shifter vidéo.

L'adresse de la base est, selon la configuration, \$78000 (520 ST), \$F8000 (1040) et \$3F8000 (Méga)

En GFA Basic :
XBIOS(2)

FONCTION 3 (\$03) : LOGBAS

Cette fonction renvoie l'adresse logique de la mémoire écran, c'est-à-dire celle sur laquelle on travaille. Bien souvent, XBIOS(2) et XBIOS(3) ont la même valeur.

En GFA Basic :
XBIOS(3)

FONCTION 4 (\$04) : GETREZ

Cette fonction renvoie la résolution de l'écran actuellement employée.

0 = Basse résolution ;

1 = Moyenne résolution ;

2 = Haute résolution.

En GFA Basic :
XBIOS(4)

FONCTION 5 (\$05) : SETSCREEN

Cette fonction permet de changer de résolution. Elle fonctionne uniquement en basse et moyenne résolution. Sous GEM, cela ne fonctionne pas, il faut donc passer les paramètres sur la pile avant l'appel à la routine VBL qui fera les modifications souhaitées. Trois paramètres sont nécessaires : les adresses logique et physique de l'écran et la résolution.

En GFA Basic :

XBIOS(5,L:adr1,L:adr2,W:n)

adr1,adr2 : mots longs ;

n : mot.

adr1 et adr2 sont des mots longs correspondant à l'adresse logique et physique et n la résolution souhaitée.

FONCTION 6 (\$06) : SETPALETTE

Cette fonction modifie tous les registres de couleurs composant la palette en une seule manipulation. On indique l'adresse de 16 mots de la nouvelle palette. Cette adresse doit être impérativement paire. L'effet est immédiat, car elle est activée dès le VBL suivant.

En GFA Basic :

XBIOS(6,L:adr)

adr : mot long.

adr pointe sur le tableau des couleurs.

FONCTION 7 (\$07) : SETCOLOR

La fonction 6 modifie tous les registres en une seule fois, la fonction 7 permet de les modifier un par un. On introduit le numéro du registre (0 à 15) et la couleur codée sous la forme \$xxx où les x correspondent aux taux de rouge, vert et bleu (0 à 7).

En GFA Basic :

XBIOS(7,W:n,W:c)

n,c : mots.

n est le numéro du registre et c, la couleur souhaitée.

FONCTION 8 (\$08) : FLOPRD

Lecture sur un ou plusieurs secteurs d'une disquette.

Voici les paramètres de la fonction :

adr : adresse du tampon qui recevra le résultat de la lecture. Cette adresse doit être impérativement paire ;

inu : inutilisé, le mettre à 0 ;

device : numéro du lecteur de disquettes A: 0 et B ;

secteur : numéro du premier secteur à lire ;

piste : numéro de la piste où commence la lecture ;

face : numéro de la face du disque (0 ou 1) ;

n : nombre de secteurs à lire.

En GFA Basic :

XBIOS(8,L:adr,W:inu,W:device,W:secteur,W:piste,W:face,W:n)

inu,device,secteur,piste,face,n : mots ;

adr : mot long.

FONCTION 9 (\$09) : FLOPWR

Ecriture sur un ou plusieurs secteurs d'une disquette.

Si l'on écrit sur le secteur BOOT (secteur 1, piste 0, face 0), le système détectera un changement de média si une fonction RWABS ou MEDIACH est invoquée.

Voici les paramètres de la fonction :

adr : adresse du tampon. Cette adresse doit être impérativement paire ;

inu : inutilisé, le mettre à 0 ;

device : numéro du lecteur de disquettes A: 0 et B: 1 ;

secteur : numéro du premier secteur à écrire ;
piste : numéro de la piste où commence l'écriture ;
face : numéro de la face du disque (0 ou 1) ;
n : nombre de secteurs à écrire.

En GFA Basic :

XBIOS(9,L:adr,W:inu,W:device,W:secteur,W:piste,W:face,W:n)

inu,device,secteur,piste,face,n : mots ;

adr : mot long.

FONCTION 10 (\$0A) : FLOPFMT

Cette fonction formate une disquette.

Voici les paramètres de la fonction :

adr : adresse du tampon d'au moins 8 Ko. Cette adresse doit être impérativement paire ;

inu : inutilisé, le mettre à 0 ;

device : numéro du lecteur de disquettes A: 0 et B: 1 ;

piste : numéro de la piste (0 à 79) ;

face : numéro de la face du disque (0 ou 1) ;

fact : facteur d'entrelacement (1 en principe) ;

magique : nombre magique 87654321 (hexadécimal) ;

valeur : cette valeur sera écrite dans chaque secteur (\$E5E5 généralement).

En GFA Basic :

XBIOS(10,L:adr,W:inu,W:device,W:piste,W:face,W:fact,L:magique,W:valeur)

inu,device,piste,face,fact,valeur : mots ;

magique,adr : mots longs.

FONCTION 11 (\$0B) : inutilisée

FONCTION 12 (\$0C) : MIDIWS

Cette fonction écrit une chaîne de caractères sur le port MIDI (MIDI OUT). On passe comme paramètres un pointeur pointant sur la chaîne caractères et le nombre de caractères moins 1 que contient cette chaîne.

En GFA Basic :

XBIOS(12,W:n,L:adr)

adr : mot long ;

n : mot.

FONCTION 13 (\$0D) : MFPOINT

Cette fonction positionne le vecteur d'interruption interne (0 à 15) du MFP à une adresse précise. On passe comme paramètres l'adresse de la routine d'interruption et le numéro de l'interruption.

En GFA Basic :

XBIOS(13,W:n,L:adr)

adr : mot long ;

n : mot.

adr correspond à l'adresse de la routine d'interruption et n au numéro de l'interruption.

FONCTION 14 (\$0E) : IOREC

Cette fonction fournit un pointeur vers un spécificateur de tampon d'un des périphériques d'entrée suivants :

0 pour le RS-232C ;

1 pour le clavier ;

2 pour l'interface MIDI IN ;

Le spécificateur renvoyé se présente de la façon suivante :

Octets 0 à 3 : adresse du tampon ;

Octets 4 et 5 : taille du tampon ;

Octets 6 et 7 : index de début (prochaine position de traitement) ;

Octet 8 et 9 : index de fin (si égal à l'index de début, le tampon est vide) ;

Octets A et B : marqueur de niveau inférieur (autorisation de poursuite de la transmission ou XON) ;

Octets C et D : marqueur de niveau supérieur (inhibition de poursuite de la transmission ou XOFF).

En GFA Basic :

XBIOS(14,W:device)

device : mot.

device correspond au périphérique en entrée.

FONCTION 15 (\$0F) : RSCONF

Cette fonction permet de configurer le port RS-232C.

En GFA Basic :

XBIOS(15,W:vit,W:mode,W:ucr,W:rsr,W:tsr,W:scr)

vit,mode,ucr,rsr,tsr,scr : mots.

vit est la vitesse de transmission en bauds :

0 --> 19200

1 --> 9600

2 --> 4800

3 --> 3600

4 --> 2400

5 --> 2000

6 --> 1800

7 --> 1200

8 --> 600

9 --> 300

A --> 200

B --> 150

C --> 134
D --> 110
E --> 75
F --> 50

mode désigne le mode de contrôle du flux :

0 --> pas de contrôle
1 --> XON/XOFF
2 --> RTS/CTS
3 --> XON/XOFF et RTS/CTS

ucr correspond à la valeur du registre UCR du MFP ;

rsr correspond à la valeur du registre RSR du MFP ;

tsr correspond à la valeur du registre TSR du MFP ;

scr correspond à la valeur du registre SCR du MFP.

FONCTION 16 (\$10) : KEYTBL

Cette fonction positionne les pointeurs vers les tables de transcodage du clavier, ce qui permet de recréer un clavier. Les paramètres sont les adresses des nouvelles tables de transcodage : table des touches normales, table des touches avec le SHIFT pressé et table avec le CAPSLOCK pressé.

En GFA Basic :

XBIOS(16,L:noshift,L:shift,L:caps)

noshift,shift,caps : mots longs

noshift est la table des touches normales ;

shift est la table des touches avec le SHIFT ;

caps est la table des touches avec le CAPSLOCK.

FONCTION 17 (\$11) : RANDOM

Cette fonction renvoie un nombre pseudo-aléatoire.

En GFA Basic :

XBIOS(17)

FONCTION 18 (\$12) : PROTOBT

Construit une image prototype du secteur BOOT (secteur 1, piste 0, face 0)

En GFA Basic :

XBIOS(18,L:adr,L:nsr,W:typd,W:exe)

adr,nsr : mots longs ;

typd,exe : mots.

adr correspond à l'adresse d'un tampon de 512 octets ;

nsr désigne le numéro de série ;
 typd est le type du disque ou -1 si ce type ne doit pas être modifié :
 0 --> 40 pistes simple face ;
 1 --> 40 pistes double face ;
 2 --> 80 pistes simple face ;
 3 --> 80 pistes double face ;
 Seuls les types 2 et 3 existent sur Atari ST.

exe est un drapeau qui indique si l'exécution du bootsecteur doit se faire :
 0 --> inexécutable ;
 1 --> exécutable.

FONCTION 19 (\$13) : FLOPVER

Cette fonction vérifie un ou plusieurs secteurs sur un disque.
 Voici les paramètres de la fonction :
 adr : adresse du tampon. Cette adresse doit être impérativement paire ;
 inu : inutilisé, le mettre à 0 ;
 device : numéro du lecteur de disquettes A: 0 et B: 1 ;
 secteur : numéro du premier secteur à vérifier ;
 piste : numéro de la piste où commence la vérification ;
 face : numéro de la face du disque (0 ou 1) ;
 n : nombre de secteurs à vérifier.

En GFA Basic :

XBIOS(19,L:adr,W:inu,W:device,W:secteur,W:piste,W:face,W:n)
 inu,device,secteur,piste,face,n : mots ;
 adr : mot long.

FONCTION 20 (\$14) : SCRDMF

Cette fonction effectue une copie d'écran sur imprimante.

En GFA Basic :

XBIOS(20)

FONCTION 21 (\$15) : CURSCONF

Cette fonction configure le curseur d'écran.

En GFA Basic :

XBIOS(21,W:code,W:n)

code,n : mots.

code indique une des fonctions suivantes :

- 0 ---> désactivation du curseur ;
 - 1 ---> activation du curseur ;
 - 2 ---> clignotement du curseur ;
 - 3 ---> fixation du curseur ;
 - 4 ---> calcul de la période de clignotement du curseur avec n ;
 - 5 ---> lecture de la période de clignotement du curseur.
- n indique le nombre de demi-cycles d'horloge (50, 60 ou 70Hz) entre chaque clignotement du curseur.

FONCTION 22 (\$16) : SETTIME

Cette fonction fixe l'heure et la date.

En GFA Basic :

XBIOS(22,L:time)

time : mot long.

time contient l'heure et la date au format DOS.

FONCTION 23 (\$17) : GETTIME

Cette fonction lit l'heure et la date.

En GFA Basic :

XBIOS(23)

FONCTION 24 (\$18) : BIOSKEYS

Cette fonction réinitialise les tables de conversion des touches du clavier.

En GFA Basic :

XBIOS(24)

FONCTION 25 (\$19) : IKDWS

Cette fonction permet l'écriture d'une chaîne de caractères sur le contrôleur intelligent clavier. On transmet en paramètre le nombre d'octets à écrire ainsi que l'adresse de la chaîne.

En GFA Basic :

XBIOS(25,W:n,L:adr)

n : mot ;

adr : mot long.

n désigne le nombre d'octets (n-1) à écrire, et adr un mot long correspondant à l'adresse de la chaîne à envoyer.

FONCTION 26 (\$1A) : JDISINT

Cette fonction empêche la validation d'une interruption de MFP 68901. Le seul paramètre correspond au numéro de l'interruption.

En GFA Basic :

XBIOS(26,W:n)

n : mot.

n est une interruption comprise entre 0 et 15.

FONCTION 27 (\$1B) : JENABINT

Cette fonction autorise la validation d'une interruption du MFP 68901. Le seul paramètre correspond au numéro de l'interruption.

En GFA Basic :

XBIOS(27,W:n)

n : mot.

n est une interruption comprise entre 0 et 15.

FONCTION 28 (\$1C) : GIACCESS

Cette fonction permet l'écriture ou la lecture d'un registre du générateur sonore. Il existe 16 registres notés de 0 à 15. On passe comme paramètre un octet contenant la donnée à écrire, suivi du numéro du registre du PSG. Le bit 7 de ce mot contiendra 0 pour une lecture et 1 pour une écriture.

En GFA Basic :

XBIOS(28,W:code,W:n)

code,n : mots.

code est la donnée à écrire et n le numéro du registre à lire ou écrire (bit 7 de n respectivement à 0 ou 1).

FONCTION 29 (\$1D) : OFFGIBIT

Cette fonction positionne à 0 un bit du port A du PSG. Le paramètre est le numéro du bit à modifier.

En GFA Basic :

XBIOS(29,W:n)

n : mot.

n est le numéro du bit à modifier.

FONCTION 30 (\$1E) : ONGIBIT

La fonction positionne à 1 un bit du port A du PSG. Le paramètre est le numéro du bit à modifier.

En GFA Basic :

XBIOS(30,W:n)

n : mot.

n est le numéro du bit à modifier.

FONCTION 31 (\$1F) : XBTIMER

Cette fonction positionne un TIMER du MFP 68901.

En GFA Basic :

XBIOS(31,W:n,W:control,W:code,L:adr)

n,control,code : mots ;

adr : mot long.

n désigne le numéro du TIMER (0-3) à modifier :

0 --> TIMER A ;

1 --> TIMER B ;

2 --> TIMER C ;

3 --> TIMER D.

control est la valeur placée dans le registre CONTROL du TIMER.

code est la valeur qui sera inscrite dans le registre de données.

adr est un pointeur vers le vecteur d'interruption.

FONCTION 32 (\$20) : DOSOUND

Cette fonction positionne le compteur sonore sur une table.

Il faut passer en paramètre un mot long qui pointe sur l'adresse de la table de commande (la commande et l'argument forment un mot ou un mot long) pour le compteur sonore.

Structure de la table :

Valeur \$00 à \$0F : indique un numéro de registre, la valeur qui suit ce nombre est à écrire dans le registre ;

Valeur \$80 : registre temporaire, la valeur qui suit ce nombre est écrite dans le registre temporaire ;

Valeur \$81 : les trois octets qui suivent ce nombre sont traités de la façon suivante : le premier indique le numéro du registre, le deuxième est la valeur à ajouter au registre temporaire en complément à deux, le troisième est la valeur terminale. Le registre temporaire est chargé dans le registre défini par le premier octet, l'opération est recommencée jusqu'à ce que le registre temporaire atteigne la valeur indiquée par le troisième octet ;

Valeur \$82 à \$FF : l'octet qui suit indique le temps à attendre avant la prochaine mise à jour. Si cette valeur vaut 0, la génération sonore est achevée.

En GFA Basic :

XBIOS(32,L:adr)

adr : mot long.

adr est l'adresse du tampon qui contient la musique à jouer.

FONCTION 33 (\$21) : SETPRT

Cette fonction permet d'indiquer le type de l'imprimante utilisée :

Bit 0 : (0) matricielle ; (1) marguerite (DAISY)

Bit 1 : (0) couleur ; (1) monochrome

Bit 2 : (0) ATARI ; (1) EPSON

Bit 3 : (0) DRAFT ; (1) TEXTE (courrier)

Bit 4 : (0) parallèle ; (1) série

Bit 5 : (0) continu ; (1) feuille à feuille

Bit 6 à bit 14 : réservé à la configuration des imprimantes

Bit 15 : mis à 0 ;

Si le mot vaut -1 (\$FFFF), la fonction renvoie l'ancienne configuration.

En GFA Basic :

XBIOS(33,code)

code : mot.

code est un mot correspondant à la configuration d'imprimante précédemment décrite.

FONCTION 34 (\$22) : KBDVBASE

Cette fonction renvoie le pointeur de la table de structure de paramètres du gestionnaire intelligent clavier.

Voici la structure du tableau des vecteurs d'interruption :

Octets 0 à 3 : adresse d'une routine d'entrée MIDI ;

Octets 4 à 7 : adresse de traitement d'une erreur clavier ;
 Octets 8 à B : adresse de traitement d'une erreur MIDI ;
 Octets C à F : adresse du gestionnaire d'état clavier ;
 Octets 10 à 13 : adresse du gestionnaire d'état souris ;
 Octets 14 à 17 : adresse du gestionnaire d'état horloge ;
 Octets 18 à 1B : adresse du gestionnaire d'état du joystick.

En GFA Basic :
XBIOS(34)

FONCTION 35 (\$23) : KBRATE

Cette fonction détermine la vitesse de répétition des touches du clavier. Deux paramètres sont passés par la pile, le délai initial avant la répétition, et la vitesse de répétition en 1/50 de seconde. Si l'un des paramètres est nul, la vitesse n'est pas modifiée. On prend sur la pile l'ancienne valeur ou la valeur actuelle si l'un des paramètres est nul. La vitesse se calcule ainsi : $DELAI * 256 + REP$.

En GFA Basic :
XBIOS(35,W:delai,W:t)

delai,t : mots.

delai désigne le délai initial avant l'auto répétition et REP la vitesse de répétition.

FONCTION 36 (\$24) : PTRBLK

Cette fonction configure l'écran pour la copie d'écran sur l'imprimante.

Le tableau de paramètres a la structure suivante :

blkprt : adresse de la mémoire vidéo ;
 offset : déplacement ;
 width : largeur ;
 height : hauteur ;
 left : pixel gauche ;
 righth : pixel droit ;
 scrres : résolution de l'écran (0-2) ;
 = 0 basse résolution ;
 = 1 moyenne résolution ;
 = 2 haute résolution ;
 dstres : qualité d'impression (0-1) ;
 = 0 épreuve ;
 = 1 définitif ;
 colpal : pointeur de la palette de couleurs ;
 typimp : type de l'imprimante (0-3) ;
 = 0 matricielle 1280 pixels par ligne ;
 = 1 matricielle couleur ;
 = 2 imprimante Atari ;
 = 3 matricielle 960 pixels par ligne ;
 port : port utilisé ;
 = 0 parallèle ;
 = 1 série RS 232C ;
 masks : table de conversion des couleurs.

En GFA Basic :
XBIOS(36,L:adr)

adr : mot long.

adr est l'adresse de la table des paramètres.

FONCTION 37 (\$25) : VSYNC

Cette fonction permet d'attendre la prochaine interruption de trame pour synchroniser l'image avec le balayage de l'écran.

En GFA Basic :
XBIOS(37)

FONCTION 38 (\$26) : SUPEREX

Cette fonction permet d'exécuter en mode superviseur une routine en langage machine logée à l'adresse passée par la pile.

En GFA Basic :
XBIOS(38,adr)

adr : mot long.

adr est l'adresse de la routine en langage machine.

FONCTION 39 (\$27) : PUNTAES

Cette fonction désactive l'AES lorsque celui-ci n'est pas en ROM dans le cas d'un fichier TOS.IMG. Cette fonction procède à un RESET machine.

En GFA Basic :
XBIOS(39)

LE GEMDOS

Le GEMDOS ressemble beaucoup au MS-DOS, le système d'exploitation des PC, à la seule différence que les paramètres sont passés par la pile et non par les registres comme en MS-DOS. L'appel au GEMDOS se fait au moyen d'une interruption TRAP (TRAP #1). Tout comme le BIOS et le XBIOS, le registre D0, du 68000, contient après l'exécution de la routine, le paramètre de retour. Le registre A0 contient le numéro de la fonction appelée (inexploitable sauf avec les instructions d'interfaçage RCALL, C:, CALL...). Voici comment appeler une fonction GEMDOS (Graphic Environment Manager System Disk Operating System) en GFA Basic :

~GEMDOS(code,par1,par2...)

ou bien alors

var=GEMDOS(code,par1,par2...)

var : variable numérique entière ;

code : expression numérique entière ;

par1,par2... : mots longs ou mots.

Cette fonction appelle la routine GEMDOS désignée par le code opération, en lui passant une liste de paramètres qui sont, soit des mots précédés par W: (pris par défaut), soit des mots longs précédés par L:. Toute erreur dans le passage des paramètres, portant sur leur valeur ou leur type (L: ou W:), peut entraîner un crash général de la machine. Au retour, la variable var contient la valeur de retour ou le code d'erreur, précédemment stockée dans D0.

FONCTION 0 (\$00) : PTERM

Cette fonction termine le processus appelé en cours et retourne au processus appelant avec un code de retour 0.

En Basic GFA :

Inutilisable.

FONCTION 1 (\$01) : CCONIN

Cette fonction permet la lecture d'un caractère en provenance du périphérique standard d'entrée avec écho (écriture) sur le périphérique standard de sortie. Le registre D0 contient le caractère lu codé de la façon suivante:

Bits 31-24 : \$00 ;

Bits 23-16 : code ou \$00 ;

Bits 15-8 : \$00 ;

Bits 7-0 : code ASCII du caractère lu.

Si le périphérique d'entrée est compatible avec Gem VDI le code des bits 23-16 sera le code de la touche enfoncée.

En Basic GFA :

GEMDOS(1)

FONCTION 2 (\$02) : CCONOUT

Cette fonction permet l'écriture d'un caractère sur le périphérique standard de sortie. Le caractère à écrire est mis sur la pile dans l'octet le moins significatif d'un mot de 16 bits, la partie haute étant à 0. Cette fonction transforme le TAB en une suite d'espaces et teste CTRL-S (arrêt défilement), CTRL-Q (reprise défilement) et CTRL-C (arrêt de l'opération en cours).

En Basic GFA :

GEMDOS(2,W:code)

code : mot.

Le paramètre code est le code ASCII du caractère à envoyer.

FONCTION 3 (\$03) : CAUXIN

Cette fonction est identique à CCONIN pour un périphérique auxiliaire : le port série. Cette instruction n'est pas très fiable.

En Basic GFA :

GEMDOS(3)

FONCTION 4 (\$04) : CAUXOUT

Cette fonction est identique à CCONOUT pour un périphérique auxiliaire : le port série.

En Basic GFA :

GEMDOS(4,W:code)

code : mot.

Le paramètre code est le code ASCII du caractère à envoyer.

FONCTION 5 (\$05) : CPRNOUT

Cette fonction écrit un caractère sur le port parallèle, c'est-à-dire sur l'imprimante.

En Basic GFA :

GEMDOS(5,W:code)

code : mot.

Le paramètre code est le code ASCII du caractère à envoyer.

FONCTION 6 (\$06) : CRAWIO

Cette fonction écrit ou lit sur la console. Un paramètre (mot de 16 bits) définit le sens, soit \$00FF pour une lecture, soit différent de \$00FF pour un affichage du caractère.

En Basic GFA :

GEMDOS(6,W:code)

code : mot.

Le paramètre code est le code à envoyer.

FONCTION 7 (\$07) : CRAWIN

Cette fonction permet la lecture d'un caractère sur le clavier sans produire d'écho sur le périphérique de sortie. Cette fonction ne teste pas les codes de contrôle.

En Basic GFA :

GEMDOS(7)

FONCTION 8 (\$08) : CNECIN

Cette fonction permet la lecture d'un caractère sur le clavier. Cette fonction teste les codes de contrôle.

En Basic GFA :

GEMDOS(8)

FONCTION 9 (\$09) : CCONWS

Cette fonction écrit une suite de caractères terminée par un octet nul sur le périphérique standard.

En Basic GFA :

GEMDOS(9,L:adr)

adr : mot long.

adr est l'adresse de la chaîne de caractères.

FONCTION 10 (\$0A) : CCONRS

Cette fonction lit une suite de caractères. Tous les codes de contrôle sont testés. Avant l'appel, le tampon doit contenir le nombre de caractères à lire dans son premier octet. Après l'appel, le tampon possède le format ci-dessous :

Octet 0 : nombre de caractères à lire, \$0D non inclus ;

Octet 1 : nombre de caractères reçus ;

Octet 2 : premier caractère reçu ;

Octet n : dernier caractère reçu, \$0D non inclus.

En Basic GFA :

GEMDOS(10,L:adr)

adr : mot long.

adr est l'adresse du tampon initialisé.

FONCTION 11 (\$0B) : CCONIS

Cette fonction détermine si un caractère est en attente sur le périphérique d'entrée standard. Le registre prendra soit la valeur \$0000 s'il n'y a pas de caractère en attente, soit \$FFFF.

En Basic GFA :

GEMDOS(11)

FONCTIONS 12-13 (\$0B-\$0C) : non utilisées

FONCTION 14 (\$0E) : DSERTDRV

Cette fonction détermine le disque qui est utilisé par défaut. Il est choisi entre les unités A (valeur 0) et P (valeur 15). Au retour, le registre D0 contient la carte des unités actives (bit à 1) dans son mot de poids faible. La carte se présente comme suit :

Bit 0 : 0 ou 1 pour disque A ;

Bit 1 : 0 ou 1 pour disque B ;

Bit 2 : 0 ou 1 pour disque C ;

...

Bit 15 : 0 ou 1 pour disque P.

En Basic GFA :

GEMDOS(14,W:n)

n : expression numérique entière.

Le paramètre n correspond à l'unité choisie.

FONCTION 15 (\$0F) : non utilisée

FONCTION 16 (\$10) : CCONOS

Cette fonction teste l'état du périphérique de sortie standard afin de déterminer s'il est prêt à recevoir un caractère. Elle admet le CTRL-C. Le registre D0 contient, soit \$00FF si le périphérique est prêt, soit \$0000 dans le cas contraire.

En Basic GFA :

GEMDOS(16)

FONCTION 17 (\$11) : CPRNOS

Cette fonction teste l'état du port parallèle, afin de déterminer s'il est prêt à recevoir un caractère. Le registre D0 contient soit \$00FF si le périphérique est prêt, soit \$0000 dans le cas contraire.

En Basic GFA :

GEMDOS(17)

FONCTION 18 (\$12) : CAUXIS

Cette fonction teste l'état du port série afin de déterminer s'il est prêt à envoyer un caractère. Le registre D0 contient soit \$00FF si le périphérique est prêt, soit \$0000 dans le cas contraire.

En Basic GFA :

GEMDOS(18)

FONCTION 19 (\$13) : CAUXOS

Cette fonction teste l'état du port série afin de déterminer s'il est prêt à recevoir un caractère. Le registre D0 contient, soit \$00FF si le périphérique est prêt, soit \$0000 dans le cas contraire.

En Basic GFA :
GEMDOS(19)

FONCTIONS 20-24 (\$14-\$18) : non utilisées

FONCTION 25 (\$19) : DGETDRIVE

Cette fonction renvoie l'unité de disque courante : 0 pour A, ..., 15 pour P.

En Basic GFA :
GEMDOS(25)

FONCTION 26 (\$1A) : FSETDTA

Cette fonction positionne l'adresse d'une table DTA (Disk Transfer Address). S'il n'y a pas de DTA positionnée, GEMDOS utilise la DTA par défaut située à l'adresse relative \$80 par rapport à la page de base (BASEPAGE+128).

En Basic GFA :
GEMDOS(26,L:adr)

adr : mot long.

adr est l'adresse de la table DTA.

FONCTIONS 27-31 (\$1B-\$1F) : non utilisées

FONCTION 32 (\$20) : SUPERVISOR

Cette fonction permet le passage du mode utilisateur en mode superviseur et vice versa. Cette fonction permet d'accéder à des variables système ou des périphériques uniquement disponibles en mode superviseur.

Nous ne la citons que par souci d'exhaustivité, car elle est inutilisable en GFA Basic.

FONCTIONS 33-41 (\$21-\$29) : non utilisées

FONCTION 42 (\$2A) : TGETDATE

Cette fonction lit la date interne. Le registre D0.W contient la date codée de la façon suivante :

Bits 0-4 : jour (1-31) ;

Bits 5-8 : mois (1-12) ;

Bits 9-F : année (0-119 que l'on ajoute à 1980).

En Basic GFA :
GEMDOS(42)

lit DATE\$, la date est renvoyée comme suit :

date = (année-1980)*512+mois*32+jour

FONCTION 43 (\$2B) : TSETDATE

Cette fonction fixe la date interne. On passe un mot de 16 bits codé de la façon suivante :

Bits 0-4 : jour (1-31) ;

Bits 5-8 : mois (1-12) ;

Bits 9-F : année (0-119 que l'on ajoute à 1980).

En Basic GFA :

GEMDOS(43,W:date)

date : mot.

le paramètre date indique la date codée comme suit :

date = (année-1980)*512+mois*32+jour

FONCTION 44 (\$2C) : TGETTIME

Cette fonction lit l'heure interne. Le registre D0.W contient codée de la façon suivante :

Bits 0-4 : secondes par pas de 2 ;

Bits 5-A : minutes (0-59) ;

Bits B-F : heures (0-23).

En Basic GFA :

GEMDOS(44)

lit TIMES\$, l'heure est renvoyée comme suit :

time = heure*2048+minute*32+seconde/2

Il faut multiplier par deux la valeur contenue dans les bits 0 à 4 pour obtenir le nombre correct de secondes.

FONCTION 45 (\$2D) : TSETTIME

Cette fonction fixe l'heure interne. On passe un mot de 16 bits codé de la façon suivante :

Bits 0-4 : secondes par pas de 2 ;

Bits 5-A : minutes (0-59) ;

Bits B-F : heures (0-23).

En Basic GFA :

GEMDOS(45,W:time)

time : mot.

time contient l'heure codée comme suit :

time = heure*2048+minute*32+seconde/2

Il faut diviser par deux le nombre correct de secondes pour obtenir la valeur à mettre dans les bits 0 à 4.

FONCTION 46 (\$2E) : non utilisée**FONCTION 47 (\$2F) : FGETDTA**

Cette fonction retourne l'adresse de la table DTA courante.

Très pratique pour les fonctions 78 et 79, FSFIRST et FSNEXT.

En Basic GFA :
GEMDOS(47)

FONCTION 48 (\$30) : SVERSION

Cette fonction renvoie le numéro de version du système d'exploitation.

En Basic GFA :
GEMDOS(48)

FONCTION 49 (\$31) : PTERMRES

Cette fonction achève le processus en cours et ne relâche pas la mémoire affectée par le processus. Le nombre d'octets à verrouiller est passé comme paramètre de la fonction. A ne pas utiliser pour des raisons de compatibilité avec les différentes versions du TOS, le GFA Basic risque de se bloquer.

En Basic GFA :
 Inutilisable.

FONCTIONS 50-53 (\$32-\$35) : non utilisées

FONCTION 54 (\$36) : DFREE

Cette fonction détermine l'espace disponible sur un disque. On passe comme paramètre l'adresse du tampon qui recevra les informations concernant l'espace disponible et le numéro de l'unité de disque (0 unité courante, 1 disque A, etc.) L'information sur l'espace libre est fournie sous la forme de 16 octets définis comme suit :

Octets 0-3 : nombre d'octets disponibles sur le disque ;

Octets 4-7 : nombre de clusters disponibles ;

Octets 8-B : nombre d'octets par secteur ;

Octets C-F : nombre de secteurs par cluster.

En Basic GFA :
GEMDOS(54,L:adr,W:n)

adr : mot long ;

n : expression numérique entière.

adr est l'adresse du tampon et la variable n est le numéro de l'unité de disque.

FONCTIONS 55-56 (\$37-\$38) : non utilisées

FONCTION 57 (\$39) : DCREATE

Cette fonction crée un sous-répertoire, c'est-à-dire un dossier. Le nom du chemin d'accès est contenu dans une chaîne de caractères terminée par \$00 à l'adresse adr.

En Basic GFA :
GEMDOS(57,L:adr)

adr : mot long.

adr est l'adresse de la chaîne de caractères déterminant le chemin d'accès.

FONCTION 58 (\$3A) : DDELETE

Cette fonction détruit un sous-répertoire, c'est-à-dire un dossier. Le nom du chemin d'accès est contenu dans une chaîne de caractères terminée par \$00 à l'adresse adr.

En Basic GFA :

GEMDOS(58,L:adr)

adr : mot long.

adr est l'adresse de la chaîne de caractères déterminant le chemin d'accès.

FONCTION 59 (\$3B) : DSETPATH

Cette fonction permet de se positionner dans un sous-répertoire, c'est-à-dire dans un dossier. Le nom du chemin d'accès est contenu dans une chaîne de caractères terminée par \$00 à l'adresse adr.

En Basic GFA :

GEMDOS(59,L:adr)

adr : mot long ;

adr est l'adresse de la chaîne de caractères déterminant le chemin d'accès.

FONCTION 60 (\$3C) : FCREATE

Cette fonction crée un fichier. Si le fichier existe déjà, la fonction retourne un code d'erreur.

Le nom du chemin d'accès est contenu dans une chaîne de caractères terminée par \$00 à l'adresse adr.

Les attributs du fichiers sont aussi passés comme paramètres.

00 = fichier en READ ONLY, lecture seule ;

01 = fichier caché, HIDDEN ;

04 = fichier système caché ;

08 = descripteur de volume, c'est-à-dire création d'un dossier.

En Basic GFA :

GEMDOS(60,L:adr,W:attribut)

adr : mot long ;

attribut : mot.

adr est l'adresse du tampon et le mot attribut représente l'attribut que l'on veut donner au fichier.

FONCTION 61 (\$3D) : FOPEN

Ouverture d'un fichier. Le nom du chemin d'accès est contenu dans une chaîne de caractères terminée par 00 à l'adresse adr. Le mode d'ouverture est aussi passé comme paramètre :

0 = fichier en READ ONLY, lecture seule ;

1 = fichier en écriture seulement ;

2 = fichier en lecture et écriture.

En Basic GFA :

GEMDOS(61,L:adr,W:mode)

adr : mot long ;

mode : mot.

adr est l'adresse du tampon et le mot mode représente le mode d'ouverture que l'on veut donner au fichier.

FONCTION 62 (\$3E) : FCLOSE

Cette fonction ferme le fichier dont le handle est spécifié en entrée. Lors de la fermeture, le répertoire et la table des allocations sont remis à jour.

En Basic GFA :

GEMDOS(62,W:handle)

handle : mot.

le paramètre handle représente le handle du fichier.

FONCTION 63 (\$3F) : FREAD

Cette fonction lit un nombre de caractères n à la position courante du fichier spécifié.

En Basic GFA :

GEMDOS(63,W:handle,L:n,L:adr)

handle : mot ;

n,adr : mots longs.

le paramètre handle représente le handle du fichier à lire, n est le nombre de caractères à lire, et adr l'adresse du tampon où seront placés ces caractères.

FONCTION 64 (\$40) : FWRITE

Cette fonction écrit un nombre de caractères n à la position courante du fichier spécifié.

En Basic GFA :

GEMDOS(64,W:handle,L:n,L:adr)

handle : mot ;

n,adr : mots longs.

le paramètre handle représente le handle du fichier à lire, n est le nombre de caractères à lire et adr l'adresse du tampon où seront lus les caractères à écrire.

FONCTION 65 (\$41) : FDELETE

Cette fonction détruit un fichier. Le nom du chemin d'accès est contenu dans une chaîne de caractères terminée par \$00 à l'adresse adr.

En Basic GFA :

GEMDOS(65,L:adr)

adr : mot long.

adr est l'adresse de la chaîne de caractères déterminant le chemin d'accès.

FONCTION 66 (\$42) : FSEEK

Cette fonction positionne le pointeur courant à l'intérieur d'un fichier. Trois paramètres sont passés par la pile : une valeur signée indiquant dans quel sens et de combien d'octets on se déplace dans le fichier (+ ou -); l'identificateur (handle) et le mode de déplacement sont définis de la manière suivante :

0 = déplacement de n octets en partant du début du fichier ;

1 = déplacement de n octets en partant de la position courante ;

2 = déplacement de n octets en partant de la fin du fichier (n doit alors forcément être négatif).

En Basic GFA :

GEMDOS(66,L:n,W:handle,W:mode)

n : mot long ;

handle,mode : mots.

n est le nombre d'octets du déplacement (signé), handle représente le handle du fichier et mode représente le mode de déplacement.

FONCTION 67 (\$43) : FATTRIB

Cette fonction détermine ou modifie les attributs d'un fichier. Le nom du chemin d'accès est contenu dans une chaîne de caractères terminée par \$00 à l'adresse adr. On indique si la fonction est en lecture ou écriture (respectivement 0 ou 1).

L'attribut du fichier est lui aussi passé en paramètre de la manière suivante :

01 = fichier en READ ONLY, lecture seule ;

02 = fichier caché, HIDDEN ;

04 = fichier système caché ;

08 = descripteur de volume ;

10 = dossier ;

20 = archive.

En Basic GFA :

GEMDOS(67,L:adr,W:op,W:attribut)

adr : mot long ;

op,attribut : mots.

adr est l'adresse du tampon, op est le numéro de l'opération et attribut représente l'attribut que l'on veut donner au fichier.

FONCTION 68 (\$44) : non utilisée**FONCTION 69 (\$45) : FDUP**

Cette fonction autorise la duplication du handle d'un fichier. Ainsi on a deux handle pour gérer un fichier. La fonction FSEEK modifie le handle et sa copie dans la même opération.

En Basic GFA :

GEMDOS(69,handle)

handle : mot.

handle est le handle du fichier qui doit être dupliqué.

FONCTION 70 (\$46) : FFORCE

Cette fonction force un descripteur de handle pour l'obliger à pointer sur le même fichier ou périphérique qu'un autre descripteur de handle.

En Basic GFA :

GEMDOS(70,W:handle1,W:handle2)

handle1,handle2 : mots.

handle1 ou handle2 sont les deux handles des fichiers.

FONCTION 71 (\$47) : DGETPATH

Cette fonction lit le chemin d'accès (PATH) actif. Il faut donner comme paramètre l'adresse d'un tampon de 64 octets qui contiendra le chemin d'accès lu. Le second paramètre correspond au numéro de l'unité de disque.

En Basic GFA :

GEMDOS(71,L:adr,W:n)

adr : mot long ;

n : mot.

adr est l'adresse du tampon et n le numéro de l'unité de disque (0 disque courant, 1 disque A, etc.).

FONCTION 72 (\$48) : MALLOC

Cette fonction permet l'allocation d'un bloc de mémoire centrale. On passe en paramètre le nombre d'octets à réserver. Si ce nombre vaut -1, le système renvoie la taille de la plus grande zone d'un seul bloc.

En Basic GFA :

GEMDOS(72,L:n)

n : mot long.

n est le nombre d'octets à réserver.

FONCTION 73 (\$49) : MFREE

Cette fonction libère une tranche de la mémoire précédemment allouée par M-MALLOC. Le paramètre est l'adresse de départ du bloc à libérer.

En Basic GFA :

GEMDOS(73,L:adr)

adr : mot long.

adr est l'adresse de départ du bloc à libérer.

FONCTION 74 (\$4A) : MSHRINK

Cette fonction diminue un espace mémoire précédemment alloué par M-MALLOC. Les paramètres sont l'adresse de départ du bloc et l'adresse du nouveau bloc.

En Basic GFA :

GEMDOS(74,W:0,L:adr,L:n)

adr,n : mots longs.

adr est l'adresse de départ du bloc et n est sa nouvelle taille.

FONCTION 75 (\$4B) : PEXEC

Cette fonction valide le chargement et l'exécution d'un processus.

En Basic GFA :

GEMDOS(75,W:op,L:adr1,L:adr2,L:adr3)

op : mot ;

adr1,adr2,adr3 : mots longs.

Voici les paramètres à passer :

op, paramètre de chargement :

1 = charge et exécute ;

3 = charge seulement ;

adr1 : adresse d'une chaîne terminée par \$00 contenant le descripteur du fichier à charger ;

adr2 : adresse du tampon contenant les informations de redirection. Cette adresse est placée à l'octet d'adresse relative \$80 (+128) par rapport à la page de base du chargement ;

adr3 : adresse d'une suite de chaînes contenant la configuration de l'environnement du processus. Les chaînes sont séparées par un octet à \$00 et terminées par deux octets à \$00.

FONCTION 76 (\$4C) : PTERM

Cette fonction termine le processus courant et retourne au processus appelant en fermant tous les fichiers ouverts.

On passe en paramètre le code de retour.

En Basic GFA :

Inutilisable.

FONCTION 77 (\$4D) : non utilisée

FONCTION 78 (\$4F) : FSFIRST

Cette fonction recherche la première entrée du dossier correspondant au nom spécifié et aux attributs spécifiés. Le nom du chemin d'accès est contenu dans une chaîne de caractères terminée par \$00 à l'adresse adr. L'attribut du fichier est lui aussi passé en paramètre de la manière suivante :

\$00 = fichier standard ;

\$01 = fichier en READ ONLY, lecture seule ;

\$02 = fichier caché, HIDDEN ;

\$04 = fichier système caché ;

\$08 = descripteur de volume ;

\$10 = dossier ;

\$20 = archive.

Lorsqu'une entrée qui correspond aux spécifications est trouvée, elle formate une DTA et renvoie cette nouvelle DTA.

En Basic GFA :

GEMDOS(78,L:adr,W:attribut)

adr : mot long ;

attribut : mot.

adr est l'adresse du tampon et attribut représente l'attribut que l'on veut rechercher.

FONCTION 79 (\$4F) : FSNEXT

Cette fonction recherche l'occurrence suivante d'un descripteur de fichier donné dans FSFIRST.

En Basic GFA :

GEMDOS(79)

FONCTIONS 80-85 (\$50-\$55) : non utilisées

FONCTION 86 (\$56) : FRENAME

Cette fonction renomme un fichier. On passe comme paramètre le descripteur du fichier original et l'adresse de la chaîne de caractères contenant le nouveau descripteur du fichier. Celle-ci est terminée par \$00.

En Basic GFA :

GEMDOS(86,W:0,L:adr1,L:adr2)

adr1,adr2 : mots longs.

FONCTION 87 (\$57) : FDATEIME

Cette fonction procède à la lecture ou l'écriture de la date et l'heure de création ou dernière modification d'un fichier. Les paramètres sont l'adresse du tampon qui contient la date et l'heure, l'handle du fichier à traiter, le type de l'opération à effectuer (0 modification, 1 lecture).

En Basic GFA :

GEMDOS(87,L:adr,W:handle,W:mode)

adr : mot long ;

handle,mode : mots.

adr est l'adresse du tampon, handle représente le descripteur du handle du fichier et mode, le type de l'opération.

ROUTINES AES ET VDI

249

La convivialité de l'Atari ST provient en grande partie des routines AES ET VDI qui simplifient la programmation des entrées-sorties en offrant une interface logicielle très puissante : Les menus et les fenêtres en sont des exemples spectaculaires.

La plupart des instructions AES et VDI sont disponibles sous une forme évoluée (surtout pour l'AES). Pour utiliser la panoplie complète des instructions et par soucis de compatibilité avec les versions antérieures, le GFA Basic 3.0 autorise les appels à travers des fonctions GEMSYS et VDISYS, après avoir convenablement initialisé les tableaux utilisés :

Pour le VDI :

CONTRL ;
INTIN ;
PTSIN ;
INTOUT ;
PTSOUT.

Pour l'AES :

GCONTRL ;
ADDRIN ;
GINTIN ;
ADDROUT ;
GINTOUT.

Tous ces tableaux sont utilisables sous la forme indicée habituelle (par exemple CONTRL(5)...) ou sous forme de tables (par exemple INT{CONTRL+10}).

Voici la liste des fonctions et des instructions du GEM traitées dans ce chapitre :

ADDRIN	OB_TYPE
ADDROUT	OB_W
APPL_EXIT	OB_X
APPL_FIND	OB_Y
APPL_INIT	OBJC_ADD
APPL_READ	OBJC_CHANGE
APPL_TPLAY	OBJC_DELETE
APPL_TRECORD	OBJC_DRAW
APPL_WRITE	OBJC_EDIT
CONTRL	OBJC_FIND
EVNT_BUTTON	OBJC_OFFSET
EVNT_DCLIC	OBJC_ORDER
EVNT_KEYBD	PTSIN
EVNT_MESAG	PTSOUT
EVNT_MOUSE	RC_COPY
EVNT_MULTI	RC_INTERSECT
EVNT_TIMER	RSRC_FREE
FORM_ALERT	RSRC_GADDR

FORM_BUTTON	RSRC_LOAD
FORM_CENTER	RSRC_OBFIX
FORM_DIAL	RSRC_SADDR
FORM_DO	SCR_P_READ
FORM_ERROR	SCR_P_WRITE
FORM_KEYBD	SHEL_ENVRN
FSEL_INPUT	SHEL_FIND
GB	SHEL_GET
GCONTRL	SHEL_PUT
GDOS?	SHEL_READ
GIN TIN	SHEL_WRITE
GIN TOUT	V~H
GRAF_DRAGBOX	VDIBASE
GRAF_GROWBOX	VDISYS
GRAF_HANDLE	V_CLRWK
GRAF_MKSTATE	V_CLSVWK
GRAF_MOUSE	V_CLSWK
GRAF_MOVEBOX	V_OPNVWK
GRAF_RUBBERBOX	V_OPNWK
GRAF_SHRINKBOX	V_CLRWK
GRAF_SLIDEBX	V_UPDWK
GRAF_WATCHBOX	VQT_EXTENT
INTIN	VQT_NAME
INTOUT	VST_LOAD_FONTS
MENU_BAR	VQT_NAME
MENU_ICHECK	VST_UNLOAD_FONTS
MENU_IENABLE	WIND_CALC
MENU_REGISTER	WIND_CLOSE
MENU_TEXT	WIND_CREATE
MENU_TNORMAL	WIND_DELETE
OB_ADR	WIND_FIND
OB_FLAGS	WIND_GET
OB_H	WIND_OPEN
OB_HEAD	WIND_SET
OB_NEXT	WIND_UPDATE
OB_SPEC	WORK_OUT
OB_STATE	
OB_TAIL	

ADDRIN

ADDRIN contient les adresses (mots longs) nécessaires à la fonction appelée.

ADDROUT

ADDROUT contient les adresses (mots longs) retournées par la fonction appelée.

APPL_EXIT()

Cette fonction provoque la sortie d'application AES initialisée grâce à APPL_INIT. Cette fonction n'est que simulée par le GFA Basic. Elle retourne 0 en cas d'erreur.

APPL_FIND(fichier)

fichier : expression alphanumérique.

Cette fonction renvoie le code d'identification d'une application dont on donne le nom de fichier (longueur obligatoire : huit caractères en majuscule, éventuellement complété par des caractères blancs). Le code est -1 en cas d'erreur.

APPL_INIT()

Cette fonction renvoie le code d'identification de l'application actuelle.

APPL_READ(code,n,adr)

Cette fonction lit n octets, dans le buffer, d'adresse adr de l'application identifiée par code.

APPL_TPLAY(adr,n,vitesse)

adr,n,vitesse : expressions numériques entières.

Réexécute les événements enregistrés par APPL_TRECORD.

La vitesse est comprise entre 1 et 10 000. L'usage de cette instruction est peu conseillé.

APPL_TRECORD(adr,n)

adr,n : expressions numériques entières.

Enregistre une séquence d'événements utilisateurs. L'usage de cette instruction est peu conseillé.

APPL_WRITE(code,n,adr)

Cette fonction écrit n octets, dans le buffer, d'adresse adr de l'application identifiée par code.

CONTRL

CONTRL contient les informations essentielles à GEM au traitement d'un appel VDI :

CONTRL(0) numéro de la fonction ;

CONTRL(1) nombres d'éléments à utiliser dans la table PTSIN ;

CONTRL(2) nombres d'éléments retournés dans la table PTSOUT ;

CONTRL(3) nombres d'éléments à utiliser dans la table INTIN ;

CONTRL(4) nombres d'éléments retournés dans la table INTOUT ;

CONTRL(5) numéro d'identification ;

CONTRL(6) numéro de la fenêtre appelée.

EVNT_BUTTON(n,masque,etat[,x,y,bouton,touche])

n,masque,etat : expressions numériques entières ;

x,y,bouton,touches : variables numériques entières.

Cette fonction paramètre et attend un événement souris. Elle permet de connaître l'état des boutons et celui du clavier en initialisant les variables lorsque l'événement attendu survient :

n : nombre de clics nécessaires ;
 masque : bit 0 : bouton gauche ;
 1 : bouton droit ;
 etat : bit 0 : bouton gauche enfoncé (si bit=1) ;
 1 : bouton droit enfoncé (si bit=1).

La fonction renvoie le nombre de clics et initialise les variables suivantes :

(x,y) : coordonnées de la souris lors de l'événement ;
 bouton : bit 0 : bouton gauche ;
 1 : bouton droit ;
 touche bit ;
 Shift droit 0 ;
 Shift gauche 1 ;
 Control 2 ;
 Alternate 3.

EVNT_DCLIC(t,code)

t,code : expressions numériques entières.

Cette fonction établit le délai t de double clic entre 0 (lent) et 4. Si code=0, la fonction renvoie l'ancien délai et ignore le nouveau ; si le code=1, la fonction valide le délai transmis.

EVNT_KEYBD()

Cette fonction attend un événement clavier et retourne un mot : code touche (octet poids faible), code caractère (octet de poids fort).

EVNT_MESAG(adr)

adr : mot long.

Cette fonction attend un message dans le tampon d'adresse adr. Si adr=0, on utilise le tampon interne du GFA Basic.

EVNT_MOUSE(code,x0,y0,largeur,hauteur,x1,y1,bouton,touche)

Cette instruction attend l'entrée ou la sortie (respectivement 0 ou 1) du curseur de la souris dans un rectangle dont le coin supérieur gauche a pour coordonnées (x0,y0) et de dimension spécifiée (largeur et hauteur). Elle initialise les variables suivantes :

(x1,y1) : coordonnées de la souris lors de l'événement ;
 bouton : bit 0 : bouton gauche ;
 1 : bouton droit ;
 touche bit ;
 Shift droit 0 ;
 Shift gauche 1 ;
 Control 2 ;
 Alternate 3.

EVNT_MULTI (type,n1,bouton,etat1,x1,y1,largeur1,hauteur1,etat2,x2,y2,largeur2,hauteur2,adr,t,[x,y,bouton,clavier,touche,n2])

Cette instruction attend divers événements. Les paramètres ont été vus dans les autres instructions EVNT.

Codage de type :

- bit : 0 Clavier ;
- 1 Bouton souris ;
- 2 Événement 1 (souris) ;
- 3 Événement 2 (souris) ;
- 4 Événement message ;
- 5 Événement temps.

Les paramètres de retour optionnels sont :

(x,y) : coordonnées de la souris lors de l'évènement ;

bouton : bouton de la souris ;

clavier : touches spéciales (Shift...) ;

touche : code touche frappée et code ASCII ;

n2 : nombre de clics effectués.

EVNT_TIMER(t)

t : expression numérique entière.

Cette fonction provoque un arrêt temporisé de t millisecondes. Elle retourne toujours 1.

FORM_ALERT(bouton,message)

bouton : expression numérique entière ;

message : expression alphanumérique.

Cette fonction édite un message (alert-box) et retourne le numéro du "bouton-texte" choisi : bouton=numéro du bouton validé par RETURN (0 -> 3); message="[n][m][b]" n=icône, m=message, b=texte-boutons (voir ALERT).

FORM_BUTTON(arbre,objet0,n,objet1)

arbre : mot long ;

objet0,n : expressions numériques entières ;

objet1 : variable numérique entière.

Cette fonction permet des entrées avec la souris. Elle renvoie 0 si un objet avec l'état EXIT à été cliqué en dernier, sinon l'entrée n'est pas encore terminée.

arbre désigne l'adresse de l'arbre, objet0 désigne l'objet lors de l'appel, n contient le nombre de clics enregistrés et objet1, le nouvel objet.

FORM_CENTER(arbre,x,y,largeur,haut)

arbre : mot long ;

x,y,largeur,haut : variables numériques entières.

Cette fonction renvoie la position d'un arbre d'objets au centre de l'écran dans les variables indiquées.

FORM_DIAL(code,x1,y1,l1,h1,x2,y2,l2,h2)

code,x1,y1,l1,h1,x2,y2,l2,h2 : expressions numériques entières.

Cette fonction gère un formulaire de dialogue dépendant du code :

0 : réserve une zone écran ;

1 : effet de rectangle en expansion ;

2 : effet de rectangle en contraction ;

4 : libère la zone réservée ;

x1,y1,l1,h1 : coordonnées minimales du rectangle ;

x2,y2,l2,h2 : coordonnées maximales du rectangle.

Elle retourne 0 en cas d'erreur.

FORM_DO(arbre,objet)

arbre : mot long ;

objet : expression numérique entière.

Cette fonction gère un formulaire jusqu'à la sortie (par clic dans un objet EXIT ou TOUCHEXIT). Elle renvoie un numéro (celui objet ayant occasionné la sortie).

FORM_ERROR(code)

code : expression numérique entière.

Cette fonction affiche un message d'erreur (codes MS-DOS). Elle renvoie le numéro du bouton validé par l'utilisateur.

FORM_KEYBD(arbre,objet1,objet2 ,car1,objet3,car2)

arbre : mot long ;

objet1,objet2 ,car1,objet3, : expressions numériques entières ;

car2 : variable numérique entière.

Cette fonction gère les entrées clavier dans un formulaire : objet1 = objet EDIT courant ;

objet2 : prochain objet EDIT ;

car1 = caractère courant ;

objet3 : objet EDIT pour l'appel suivant ;

car2 : caractère suivant.

Elle retourne 0 si un objet EXIT a été sélectionné.

FSEL_INPUT(repertoire,fichier,[bouton])

repertoire,fichier : expressions alphanumériques ;

bouton : variable numérique entière.

Cette fonction appelle le sélecteur de fichier standard. repertoire indique le chemin des répertoires présélectionné, et fichier présélectionnés. Au retour, la variable bouton contient le bouton choisi : 0 ou 1, respectivement pour "Arrêt" ou "Ok".

GB

La table GB contient les adresses des tables:

GB+0 contient l'adresse de GCONTRL ;

GB+8 contient l'adresse de GINTIN ;

GB+12 contient l'adresse de GINTOUT ;

GB+16 contient l'adresse de ADDRIN ;

GB+20 contient l'adresse de ADDRROUT.

Son intérêt est succinct puisque le GFA Basic reconnaît les tables **GCONTRL**, **GIN TIN**, **GIN TOUT**, **ADDRIN**, et **ADDRROUT**. De plus, elle est la seule table qui n'est pas accessible avec des indices.

GCONTRL

GCONTRL contient les informations essentielles à GEM au traitement d'un appel AES :

GCONTRL(0) numéro de la fonction ;

GCONTRL(1) taille de la table **GIN TIN** ;

GCONTRL(2) taille de la table **GIN TOUT** ;

GCONTRL(3) taille de la table **ADDRIN** ;

GCONTRL(4) taille de la table **ADDRROUT**.

GDOS?

Cette fonction booléenne renvoie **TRUE** si **GDOS** est chargé et **FALSE** dans le cas contraire.

GEMSYS n

n : expression numérique entière.

Cette instruction appelle la routine AES dont le numéro est **n**.

GIN TIN

GIN TIN contient les informations passées en paramètres (mots) avant l'appel, elles diffèrent suivant la fonction AES appelée.

GIN TOUT

GIN TOUT contient les informations retournées (mots) par la fonction après l'appel, elles diffèrent suivant la fonction AES appelée.

GRAF_DRAGBOX(largeur1,hauteur1,x1,y1,largeur0,hauteur0,x0,y0[,x2,y3])

largeur1,hauteur1,x1,y1,largeur0,hauteur0,x0,y0 : expressions numériques entières ;

x2,y3 : variables numériques entières.

Cette fonction gère le déplacement d'un rectangle (largeur1, hauteur1, x1, y1) par l'utilisateur à l'intérieur d'un cadre (largeur0, hauteur0, x0, y0). Les variables x2 et y3 contiennent les nouvelles coordonnées du rectangle quand l'utilisateur relâche le bouton. Elle renvoie 0 en cas d'erreur.

GRAF_GROWBOX(x1,y1,largeur1,hauteur1,x2,y2,largeur2,hauteur2)

x1,y1,largeur1,hauteur1,x2,y2,largeur2,hauteur2 : expressions numériques entières ;

Figure l'expansion d'un rectangle qui passe de l'état 1 à l'état 2. Elle renvoie 0 en cas d'erreur.

GRAF_HANDLE(largeur,hauteur,largeur0,hauteur0)

largeur,hauteur,largeur0,hauteur0 : expressions numériques entières ;
 Cette fonction retourne le code station VDI. Elle initialise également les dimensions (pixels) des caractères (largeur,hauteur) et celles du rectangle d'inscription(largeur0,hauteur0).

GRAF_MKSTATE(x,y,bouton,clavier)

x,y,bouton,clavier : variables numériques entières.

Cette fonction indique les coordonnées de la souris et l'état du clavier. Le bit 0 de bouton donne l'état du bouton gauche, le bit 1 l'état du bouton droit. L'état du clavier suppose un test de bit :

1 : Shift droit ;

2 : Shift gauche ;

4 : Control ;

8 : Alternate.

La fonction retourne toujours 1.

GRAF_MOUSE(code,adr)

adr : mot long ;

code : expression numérique entière.

code indique la forme de la souris et adr l'adresse de la table des motifs. Cette fonction est équivalente de DEFMOUSE. Elle retourne 0 en cas d'erreur.

GRAF_MOVEBOX(largeur,hauteur,x0,y0,x1,y1)

largeur,hauteur,x0,y0,x1,y1 : expressions numériques entières.

Cette fonction déplace un rectangle à l'écran de la position 1 à la position 2. Elle retourne 0 en cas d'erreur.

GRAF_RUBBERBOX(x,y,largeur0,hauteur0[,largeur1,hauteur1])

largeur1,hauteur1 : variables numériques entières ;

x,y,largeur0,hauteur0 : expressions numériques entières.

Cette fonction permet d'étirer un rectangle avec la souris. les variables largeur1,hauteur1 contiennent la largeur et la hauteur du rectangle après l'arrêt de la fonction. Elle retourne 0 en cas d'erreur.

GRAF_SHRINKBOX(x0,y0,l0,h0,x1,y1,l1,h1)

l0,h0,l1,h1,x0,y0,x1,y1 : expressions numériques entières.

Cette fonction dessine un rectangle en contraction de l'état 0 à l'état 1. Elle retourne 0 en cas d'erreur.

GRAF_SLIDEBOX(arbre,objet,poussoir,sens)

arbre : mot long ;

objet,poussoir,sens : expressions numériques entières.

Cette fonction déplace un "ascenseur graphique" (bouton-poussoir dans son objet parent) dans le sens horizontal (0) ou vertical (1). Elle retourne la position relative, comprise entre 0 et 1000.

GRAF_WATCHBOX(arbre,objet,etat0,etat1)

arbre : mot long ;

objet,etat0,etat1 : expressions numériques entières.

Cette fonction change l'état de l'objet si la souris entre ou sort de sa zone graphique. Elle retourne 0 (dehors) ou 1 (dedans).

INTIN

INTIN contient des paramètres dont la longueur est sujette à des variations nécessaires à la fonction appelée.

INTOUT

INTOUT contient une série de paramètres retournés par la fonction appelée.

MENU_BAR(arbre,code)

arbre : mot long ;

code : expression numérique entière.

Cette fonction active (code=1) ou désactive (0) la barre de menu. Elle retourne 0 en cas d'erreur.

MENU_ICHECK(arbre,n,code)

arbre : mot long ;

n,code : expressions numériques entières.

Cette fonction coche un l'item du menu de numéro n si code=0 ou l'enlève si code=1. Elle retourne 0 si une erreur est apparue.

MENU_IENABLE(arbre,n,code)

arbre : mot long ;

n,code : expressions numériques entières.

Cette fonction édite en grisé (code=0) ou en mode normal (1) un item du menu. Elle retourne 0 en cas d'erreur.

MENU_REGISTER(id,titre)

id : expression numérique entière ;

titre : expression alphanumérique.

Cette fonction place le titre de l'accessoire sous le menu "Bureau". Elle retourne le code de l'accessoire (0 -> 5) ou -1 en cas d'erreur.

MENU_TEXT(arbre,n,texte)

arbre : mot long ;

n : expression numérique entière ;

texte : expression alphanumérique.

Cette fonction change le texte de l'item du menu portant le numéro n. Elle retourne 0 en cas d'erreur.

MENU_TNORMAL(arbre,n,code)

arbre : mot long ;

n,code : expressions numériques entières.

Cette fonction met le titre, portant le numéro *n*, en mode normal (1) ou inverse vidéo (0). Elle retourne 0 en cas d'erreur.

OB_ADR(arbre,objet)

arbre : mot long ;

objet : expression numérique entière.

Cette fonction donne l'adresse de l'objet dont le numéro est spécifié.

OB_FLAGS(arbre,objet)

arbre : mot long ;

objet : expression numérique entière.

Cette fonction renvoie l'état des flags de l'objet. Effectuer un test sur les bits pour déterminer l'état :

0 : SELECTABLE ;

1 : DEFAULT ;

2 : EXIT ;

3 : EDITABLE ;

4 : RBUTTON ;

5 : LASTOB ;

6 : TOUCHEXIT ;

7 : HIDETREE ;

8 : INDIRECT.

OB_H(arbre,objet)

arbre : mot long ;

objet : expression numérique entière.

Cette fonction renvoie la hauteur de l'objet.

OB_HEAD(arbre,objet) arbre : mot long ;

objet : expression numérique entière.

Cette fonction renvoie le numéro du premier objet fils de l'arbre.

OB_NEXT(arbre,objet)

arbre : mot long ;

objet : expression numérique entière.

Cette fonction donne le numéro de l'objet suivant (de même niveau) ou de l'objet père (dernier élément du niveau).

OB_SPEC(arbre,objet)

arbre : mot long ;

objet : expression numérique entière.

Mot long (fonction du type de l'objet). Cette fonction renvoie un pointeur sur les structures d'information. Voir également OB_TYPE.

20 : G_BOX Couleur et épaisseur de ligne ;

21 : G_TEXT Adresse structure TEDINFO ;

22 : G_BOXTEXT Adresse structure TEDINFO ;

23 : G_IMAGE	Adresse structure BITBLK ;
24 : G_USERDEF	Adresse structure USERBLK ;
25 : G_IBOX	Couleur, épaisseur (voir plus bas) ;
26 : G_BUTTON	Adresse chaîne texte ;
27 : G_BOXCHAR	(voir plus bas) ;
28 : G_STRING	Adresse de chaîne ;
29 : G_FTEX	Adresse structure TEDINFO ;
30 : G_FBOXTEXT	Adresse structure TEDINFO ;
31 : G_ICON	Adresse structure ICONBLK ;
32 : G_TITLE	Adresse de chaîne.

Pour G_BOXCHAR, les huit bits supérieurs sont définis et contiennent le code ASCII du caractère. Pour G_BOXCHAR, G_BOX et G_IBOX, la définition de la couleur est codée sur quatre bits :

Format de bits : 11112222 34445555

- 1 : cadre (de 0 à 15) ;
- 2 : texte (de 0 à 15) ;
- 3 : mode texte (0 : transparent, 1 : normal) ;
- 4 : motif trame (de 0 à 7) ;
- 5 : couleur intérieure de l'objet.

Le cadre est codé suivant la convention :

- 0 : absence de cadre.
- 1 -> 128 : bord du cadre situé à n pixels à l'intérieur de l'objet ;
- 128 -> -1 : bord du cadre situé à n pixels à l'extérieur de l'objet.

OB_STATE(arbre,objet)

arbre : mot long ;

objet : expression numérique entière.

Cette fonction renvoie l'état de l'objet codé sur 5 bits :

bit

- 0 : SELECTED ;
- 1 : CROSSED ;
- 2 : CHECKED ;
- 3 : DISABLED ;
- 4 : OUTLINED ;
- 5 : SHADOWED.

OB_TAIL(arbre,objet)

arbre : mot long ;

objet : expression numérique entière. Cette fonction renvoie le numéro dernier objet fils.

OB_TYPE(arbre,objet)

arbre : mot long ;

objet : expression numérique entière. Cette fonction renvoie le type de l'objet

de numéro spécifié. Voir également OB_SPEC.

20 : G_BOX ;
 21 : G_TEXT ;
 22 : G_BOXTEXT ;
 23 : G_IMAGE ;
 24 : G_USERDEF ;
 25 : G_IBOX ;
 26 : G_BUTTON ;
 27 : G_BOXCHAR ;
 28 : G_STRING ;
 29 : G_FTEXT ;
 30 : G_FBOXTEXT ;
 31 : G_ICON ;
 32 : G_TITLE.

OB_W(arbre,objet)

arbre : mot long ;
 objet : expression numérique entière.
 Cette fonction renvoie la largeur de l'objet.

OB_X(arbre,objet)

arbre : mot long ;
 objet : expression numérique entière.
 Cette fonction renvoie l'abscisse de l'objet.

OB_Y(arbre,objet)

arbre : mot long ;
 objet : expression numérique entière.
 Cette fonction renvoie l'ordonnée de l'objet.

OBJC_ADD(arbre,objet0,objet1)

objet0,objet1 : expressions numériques entières ;
 arbre : mot long.
 Cette fonction ajoute un objet à l'arbre. objet 0 est le numéro de l'objet-père et objet1, l'objet-fils à ajouter. Elle retourne 0 en cas d'erreur.

OBJC_CHANGE(arbre,obj,0,x,y,largeur,hauteur,état,code)

arbre : mot long ;
 obj,0,x,y,largeur,hauteur,état,code : expressions numériques entières.
 Cette fonction change l'état de l'objet et le redessine (code=1) ou non (0) dans la zone délimitée. Voir également OB_STATE pour le codage de l'état. Elle retourne 0 en cas d'erreur.

OBJC_DELETE(arbre,objet)

arbre : mot long ;
 objet : expression numérique entière.

Cette fonction supprime l'objet. Elle retourne 0 en cas d'erreur.

OBJC_DRAW(arbre,objet0,n,x,y,largeur,hauteur)

arbre : mot long ;

objet0,n,x,y,largeur,hauteur : expressions numériques entières.

Cette fonction dessine l'objet (objet0) et les objets suivants (sur n niveaux d'objets) dans la zone spécifiée. Elle retourne 0 en cas d'erreur.

OBJC_EDIT(arbre,objet,car,position,code,curseur)

arbre : mot long ;

objet,car,position,code : expressions numériques entières ;

curseur : variable numérique entière.

Cette fonction édite un caractère dans des objets de type G_BOXTEXT et G_TEXT et le positionne dans la chaîne d'entrée. Au retour, la variable curseur contient la nouvelle position du caractère.

code :

1 : affiche curseur, modifie contenu des chaînes ;

2 : affiche caractère valide ;

3 : supprime curseur.

La fonction renvoie 0 en cas d'erreur.

OBJC_FIND(arbre,objet0,n,x,y)

arbre : mot long ;

objet0,n : expressions numériques entières ;

x,y : variables numériques entières.

Elle retourne l'objet situé sous les coordonnées (x,y), avec une profondeur donnée n, en commençant à l'objet dont le numéro est objet0. Elle retourne -1 en cas d'échec.

OBJC_OFFSET(arbre,objet,x,y)

arbre : mot long ;

objet : expression numérique entière ;

x,y : variables numériques entières.

Cette fonction calcule les coordonnées absolues de l'objet. Elle retourne 0 en cas d'erreur.

OBJC_ORDER(arbre,objet,n)

arbre : mot long ;

objet,n : expressions numériques entières.

Cette fonction change le niveau n de l'objet.

-1 : premier niveau ;

0 : dernier ;

1 : avant-dernier,...

Elle renvoie 0 en cas d'erreur.

PTSIN

PTSIN contient les coordonnées passées en paramètres (mots) avant l'appel elles diffèrent suivant la fonction VDI appelée.

PTSOUT

PTSOUT contient les coordonnées ou les valeurs retournées (mots) par la fonction après l'appel elles diffèrent suivant la fonction VDI appelée.

RC_COPY adr1,x,y,largeur,hauteur TO adr2,x1,y1[,mode]

adr1,adr2 : mots longs ;

x,y,largeur,hauteur,x1,y1,mode : expressions numériques entières.

Cette instruction copie un rectangle de la page-écran pointée par adr1 sur la page-écran pointée par adr2. Les nouvelles coordonnées dans la nouvelle page sont (x1,y1). mode indique le mode de copie (0->15) identique à celui de PUT.

RC_INTERSECT(x1,y1,largeur1,hauteur1,x2,y2,largeur2,hauteur2)

x1,y1,largeur1,hauteur1,x2,y2,largeur2,hauteur2) : expressions numériques entières.

Elle retourne TRUE (-1) si les rectangles 1 et 2 ont une intersection (la surface délimitée est mise dans x2, y2, largeur2, hauteur2), ou FALSE (0) dans le cas contraire.

RSRC_GADDR(type,objet,adr)

code,objet : expressions numériques entières ;

adr : variable numérique entière.

Cette fonction donne l'adresse de la structure du type donné, de l'objet spécifié.

Type de la structure :

0 : arbre ;

1 : OBJECT ;

2 : TEDINFO ;

3 : ICONBLK ;

4 : BITBLK ;

5 : STRING ;

6 : image ;

7 : obspec ;

8 : te_ptext ;

9 : te_ptmplt ;

10 : te_pvalid ;

11 : ib_pmask ;

12 : ib_pdata ;

13 : ib_ptext ;

14 : bi_pdata ;

15 : ad_frstr ;

16 : ad_frimg.

Elle retourne 0 en cas d'erreur.

RSRC_FREE()

Cette fonction libère la zone allouée à un fichier ressource par **RSRC_LOAD**. Elle retourne 0 en cas d'erreur.

RSRC_LOAD(fichier)

fichier : expression alphanumérique.

Cette fonction charge un fichier ressource, avec allocation de l'espace mémoire nécessaire. Elle retourne 0 en cas d'erreur.

RSRC_OBFIX(arbre,objet)

arbre : mot long ;

objet : expression numérique entière.

Cette fonction opère la conversion des coordonnées objet en pixels.

RSRC_SADDR(type,objet,adr)

type,objet : expressions numériques entières ;

adr : variable numérique entière.

Cette fonction stocke l'adresse de la structure de type donné (voir **RSRC_GADDR**), de l'objet spécifié. Elle retourne 0 en cas d'erreur.

SCR_P_READ(chemin)

chemin : variable alphanumérique.

Cette fonction lit le nom du fichier stocké dans le tampon. Elle retourne 0 en cas d'erreur.

SCR_P_WRITE(chemin)

chemin : variable alphanumérique.

Cette fonction écrit le nom du fichier dans le tampon. Elle retourne 0 en cas d'erreur.

SHEL_ENVRN(adr,chaine)

adr : variable numérique entière ;

chaine : expression alphanumérique.

Cette fonction recherche une chaîne dans le tampon de l'AES.

SHEL_FIND(chemin)

chemin : variable alphanumérique.

Cette fonction recherche un fichier, elle retourne le chemin complet si le fichier est trouvé, et 0 en cas d'erreur.

SHEL_GET(n,chaine)

n : expression numérique entière ;

chaine : variable alphanumérique.

Cette fonction lit n octets dans la mémoire du DESKTOP.INF. Elle retourne 0 en cas d'erreur.

SHEL_PUT(n,ch\$)

Cette fonction écrit n octets dans le tampon du DESKTOP.INF. Elle retourne 0 en cas d'erreur.

SHEL_READ(commande,nom)

commande,nom : variables alphanumériques.

Cette fonction lit la ligne de commandes et son propre nom. Elle retourne 0 en cas d'erreur.

SHEL_WRITE(prog,tos,gem,commande,nom)

prog,tos : expressions numériques entières ;

commande,nom : expressions alphanumériques.

Cette fonction lance une application avec son nom et sa ligne de commandes. le programme résidant est effacé.

prog : 0=retourner au bureau ;

1=charger le programme ;

tos : 0=programme TOS ;

1=programme graphique ;

gem : 0=pas un programme GEM ;

1=programme GEM.

Elle retourne 0 en cas d'erreur.

V~H

Cette variable contient le handle VDI interne au GFA Basic. Il possible de fixer cette variable sur une valeur quelconque par simple affectation.

VDIBASE

Cette variable contient l'adresse de base de la mémoire VDI.

VDISYS [code[,n1,n2[,souscode]]]

code,n1,n2,souscode : expressions numériques entières.

Cette instruction appelle la routine AES spécifiée par code. Les expressions facultatives n1, n2 et souscode indiquent respectivement le nombre de valeurs dans le tableau entier d'entrée, le nombre de valeurs dans le tableau de coordonnées de points et le code d'une sous-routine VDI à appeler.

V_CLRWK()

Cette fonction vide le tampon de sortie.

V_CLSVWK(id)

id : expression numérique entière.

Cette fonction ferme une station de travail virtuelle dont l'identificateur est id.

V_CLSWK()

Cette fonction ferme la station de travail courante.

V_OPNVWK(id,v1,v2,v3...)

id,v1,v2,v3... : expressions numériques entières.

Si le GDOS est présent, Cette fonction ouvre un écran virtuel avec paramétrage éventuel, et retourne le handle correspondant à l'identificateur du périphérique transmis. Voir également l'instruction V_OPNWK.

V_OPNWK(id,v1,v2,v3,...)

id,v1,v2,v3... : expressions numériques entières.

Si le GDOS est présent, cette fonction ouvre une station de travail, avec un éventuel paramétrage et retourne le handle correspondant à l'identificateur de périphérique spécifié (mot) :

Ecran	1 ;
Plotter	11 ;
Imprimante	21 ;
Fichier META	31 ;
Caméra	41 ;
Tableur graph.	51.

V_UPDWK()

Cette fonction envoie sur le périphérique concerné les instructions graphiques stockées dans le tampon, avec une station ouverte par V_OPNWK().

VQT_EXTENT(texte[,x1,y1,x2,y2,x3,y3,x4,y4])

texte : : expression alphanumérique ;

x1,y1,x2,y2,x3,y3,x4,y4 : variables numériques entières.

Si le GDOS est présent, cette fonction donne les coordonnées du rectangle dans lequel le texte transmis s'inscrit. Les coordonnées sont placées de PTSOUT(0) à PTSOUT(7) ou dans les variables transmises :

(x1,y1) : coin inférieur gauche ;
 (x2,y2) : coin inférieur droit ;
 (x3,y3) : coin supérieur droit ;
 (x4,y4) : coin supérieur gauche.

VQT_NAME(id,nom)

id : variable numérique entière ;

nom : variable alphanumérique.

Si le GDOS est présent, cette fonction donne l'identificateur de la police chargée avec VST_LOAD_FONTS() et son nom.

VST_LOAD_FONTS(0)

Si le GDOS est présent, cette fonction charge les jeux de caractères contenus dans ASSIGN.SYS et retourne le nombre de jeux effectivement chargés (éventuellement nul).

VST_UNLOAD_FONTS(0)

Si le GDOS est présent, cette fonction élimine les jeux de caractères chargés avec VST_LOAD_FONTS().

WIND_CALC(type,attribut,x1,y1,largeur1,hauteur1,x2,y2,largeur2,hauteur2)
 type,attribut,x1,y1,largeur1,hauteur1 : expressions numériques entières ;
 x2,y2,largeur2,hauteur2 : variables numériques entières.

Cette fonction calcule la taille de la zone de travail d'une fenêtre à partir de ses dimensions totales (code=1) ou le contraire. Elle calcule les dimensions totales connaissant les dimensions de la zone de travail (code=0). Les coordonnées : x2, y2, largeur2, hauteur2 contiennent les résultats. Le codage des attributs de la fenêtre (attribut) est donné dans WIND_CREATE. Elle retourne 0 en cas d'erreur.

WIND_CLOSE(handle)

handle : expression numérique entière.

Cette fonction referme la fenêtre sans la détruire. Voir également la fonction WIND_DELETE.

Elle retourne 0 en cas d'erreur.

WIND_CREATE(attribut,x,y,largeur,hauteur)

attribut,x,y,largeur,hauteur : expressions numériques entières.

Cette fonction crée une fenêtre sans l'afficher. Le paramétrage des composants de la fenêtre (attribut) est le suivant :

Bit	Hexa	
0	&H0001	NAME Titre ;
1	&H0002	CLOSE Case de fermeture ;
2	&H0004	FULL Case de taille maximale ;
3	&H0008	MOVE Barre de déplacement ;
4	&H0010	INFO Ligne d'information ;
5	&H0020	SIZE Case de redimensionnement ;
6	&H0040	UPARROW Flèche supérieure ;
7	&H0080	DNARROW Flèche inférieure ;
8	&H0100	VSLIDE Ascenseur vertical ;
9	&H0200	LFARROW Flèche gauche ;
10	&H0400	RTARROW Flèche droite ;
11	&H0800	HSLIDE Ascenseur horizontal.

Il suffit d'additionner les codes pour obtenir un paramétrage intégrant plusieurs composants de fenêtre. Elle retourne l'identificateur (handle) de la fenêtre, une valeur négative s'il n'y a plus de fenêtre et 0 en cas d'erreur.

WIND_DELETE(handle)

handle : expression numérique entière.

Cette fonction supprime une fenêtre et récupère l'espace mémoire.

Elle retourne 0 en cas d'erreur.

WIND_FIND(x,y)

x,y : expressions numériques entières.

Elle retourne le handle de la fenêtre située sous les coordonnées spécifiées, et 0 s'il n'y en a pas.

WIND_GET(handle,code,x,y,largeur,hauteur)

x,y,largeur,hauteur : variables numériques entières ;

handle,code : expression numérique entière.

Cette fonction permet d'obtenir des informations concernant le paramétrage des composants d'une fenêtre. En fonction du code transmis, les variables x, y, largeur et hauteur sont différemment affectées :

code :

- 4 Coordonnées de la zone de travail (WF_WORKXYWH) ;
- 5 Coordonnées de la fenêtre entière (WF_CURRXYWH) ;
- 6 Coordonnées de la fenêtre entière précédente (WF_PREVXYWH) ;
- 7 Coordonnées de la plus grande fenêtre possible (WF_FULLXYWH) ;
- 11 Coordonnées du premier rectangle de retraçage (WF_FIRSTXYWH) ;
- 12 Coordonnées du rectangle de retraçage suivant (WF_NEXTXYWH).

Dans ces six cas (code=4, 5, 6, 7, 11, 12), les variables contiennent :

x : Abscisse (x) du coin supérieur gauche ;

y : Ordonnée (y) du coin inférieur gauche ;

largeur : largeur de la zone ou de la fenêtre ;

hauteur : hauteur de la zone ou de la fenêtre.

8 x contient la position (1=bord gauche, 1000=bord droit) de l'ascenseur horizontal (WF_HSLIDE) ;

9 x contient la position (1=en haut, 1000=tout en bas) de l'ascenseur vertical (WF_VSLIDE) ;

10 x contient l'identificateur de la fenêtre active (WF_TOP) ;

13 Réserve ;

15 Dimension de l'ascenseur horizontal par rapport à la barre de défilement : x=-1 (taille minimale) ou compris entre 1 (petite), et 1000 (WF_HSLIZE) ;

16 Dimension de l'ascenseur vertical par rapport à la barre de défilement : x=-1 (taille minimale) ou compris entre 1 (petite) et 1000 (WF_VSLIZE).

WIND_OPEN(handle,x,y,largeur,hauteur)

handle,x,y,largeur,hauteur : expressions numériques entières. Cette fonction affiche la fenêtre précédemment créée par WIND_CREATE. Elle retourne 0 en cas d'erreur.

WIND_SET(handle,code,x,y,r1,r2)

handle,code : expressions numériques entières ;

x,y,r1,r2 : variables numériques entières.

Cette fonction modifie le paramétrage des composants de la fenêtre spécifiée ; en fonction du code, les variables x, y, r1 et r2 sont différemment affectées :

1 Choix des composants de la fenêtre (WF_KIND). x=valeur résultante (voir WIND_CREATE) ;

2 Nouveau titre (WF_NAME). Adresse de la chaîne dans x et y ;

3 Nouvelle ligne d'information (WF_INFO). Adresse de la chaîne dans x et y ;

5 WF_CURRXYWH : cf WIND_GET ;

8 WF_HSLIDE : cf WIND_GET ;

9 WF_VSLIDE : cf WIND_GET ;
 10 WF_TOP : cf WIND_GET ;
 15 WF_HSLIZE : cf WIND_GET ;
 16 WF_VSLIZE : cf WIND_GET ;
 14 Nouvel arbre de menu Bureau (WF_NEWDESK). Elle place l'octet fort de l'adresse dans x, l'octet faible dans y et le numéro du premier objet à dessiner dans r1.

WIND_UPDATE(code)

Cette fonction indique à l'AES que l'application :

- 0 : a terminé de retracer la fenêtre (END_UPDATE) ;
- 1 : débute l'opération de traçage (BEG_UPDATE) ;
- 2 : donne le contrôle de la souris à l'AES (END_MCTRL) ;
- 3 : l'application gère totalement la souris, l'AES ne gère plus les menus (BEG_MCTRL).

WORK_OUT(indice)

indice : expression numérique entière.

Cette fonction retourne une valeur contenue dans le tableau des paramètres du VDI (ind : 0 -> 56).

ANNEXES

271

TABLE DES CODES ASCII

Poids faible

	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E
2	:	"	#	\$	%	&	'	()	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N
5	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~
8	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>
9	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>
A	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>
B	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>
C	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>
D	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>
E	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>
F	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>

Poids fort

Esc
TABLE DES CODES ESCAPE

P chr \$ (27); "Lettre"

Les séquences escape de l'émulateur VT 52 peuvent être reprises avec les instructions PRINT, INPUT et OUT dans le GFA Basic.

En voici une liste exhaustive :

ESC A

Le curseur monte d'une ligne (il se positionne sur la case immédiatement supérieure), sauf s'il se trouve déjà sur la première ligne.

ESC B

Le curseur descend d'une ligne (il se positionne sur la case immédiatement inférieure), sauf s'il se trouve déjà sur la dernière ligne.

ESC C

Le curseur se déplace d'une colonne vers la gauche, sauf s'il se trouve déjà sur la dernière colonne.

ESC D

Le curseur se déplace d'une colonne vers la droite, sauf s'il se trouve déjà sur la première colonne.

ESC E

Effacement de l'écran avec positionnement dans le coin supérieur gauche.

ESC H

Le curseur se positionne dans le coin supérieur gauche sans rafraîchir l'écran.

ESC I

Commande similaire à ESC A, à la seule différence que le curseur, s'il se trouve sur la première ligne, insère une ligne (scrolling vers le bas). La dernière ligne disparaît.

ESC J

Effacement de l'écran à partir de la position du curseur.

ESC K

Effacement de la ligne à partir de la position du curseur. Le caractère sous le curseur est perdu.

ESC L

Insertion d'une ligne au-dessus de la position du curseur. Ce dernier se place au début de la nouvelle ligne. La dernière ligne disparaît (scrolling vers le bas).

ESC M

Effacement de la ligne du curseur; celui-ci se place alors au début de la ligne suivante (scrolling vers le haut).

ESC Y colonne+32 ligne+32

Le curseur se positionne dans la case de coordonnées (colonne,ligne).

ESC b chiffre

Sélection de la couleur du crayon. Le paramètre chiffre est fonction de la résolution utilisée : 0 ou 1 en monochrome, 0 à 3 en moyenne résolution, 0 à 15 en basse résolution.

ESC c chiffre

Sélection de la couleur de fond. Le paramètre se comporte de la même manière que précédemment.

ESC d

Effacement de l'écran jusqu'à la position du curseur.

ESC e

Le curseur texte devient visible; il s'agit d'un rectangle de la couleur d'écriture qui clignote.

ESC f

Le curseur disparaît.

ESC j

Mémorisation de la position courante du curseur afin de pouvoir le repositionner si besoin est.

ESC k

Positionnement du curseur à l'endroit mémorisé.

ESC l

Effacement de la ligne du curseur; celui-ci se place alors au début de la ligne détruite (pas de scrolling).

ESC o

Effacement de la ligne du curseur jusqu'à celui-ci.

ESC p

Passage en mode vidéo inverse.

ESC q

Retour au mode vidéo normal.

ESC v

Passage en mode d'insertion automatique de ligne. Dès que le curseur arrive en fin de ligne, une nouvelle ligne s'insère.

ESC w

Annulation du mode d'insertion automatique de ligne.

RECAPITULATIF DES DEFINITIONS

DEFBIT ch\$

ch\$: expression alphanumérique.

Cette instruction définit une ou plusieurs variables booléennes.

DEFBYT ch\$

ch\$: expression alphanumérique.

Cette instruction définit une ou plusieurs variables entières codées sur un octet.

DEFFLT ch\$

ch\$: expression alphanumérique.

Cette instruction définit une ou plusieurs variables en virgule flottante codées sur huit octets.

DEFINT ch\$

ch\$: expression alphanumérique.

Cette instruction définit une ou plusieurs variables entières signées codées sur quatre octets, c'est-à-dire un mot long.

DEFLIST n

n : expression numérique entière.

Cette instruction définit le mode d'affichage des variables, des procédures et des fonctions.

0 --> les mots clefs du GFA Basic sont affichés en majuscules, les noms des variables, des procédures et des fonctions utilisateurs sont affichés en minuscules ;

1 --> les mots clefs du GFA Basic, les noms des variables, des procédures et des fonctions utilisateurs sont affichés avec leur première lettre en majuscules et le reste en minuscules ;

2 --> les mots clefs du GFA Basic sont affichés en majuscules, les noms des variables, des procédures et des fonctions utilisateurs sont affichés en minuscules avec le suffixe correspondant au type ;

3 --> les mots clefs du GFA Basic, les noms des variables, des procédures et des fonctions utilisateurs sont affichés avec leur première lettre en majuscules et le reste en minuscules. Le suffixe correspondant au type est placé par le GFA Basic.

DEFSTR ch\$

ch\$: expression alphanumérique.

Cette instruction définit une ou plusieurs variables alphanumériques.

DEFWRD ch\$

ch\$: expression alphanumérique.

Cette instruction définit une ou plusieurs variables entières signées codées sur deux octets, c'est-à-dire un mot.

DEFFILL [c] [,a] [,b]

DEFFILL [c,] var\$

a,b,c : expressions numériques entières ;

var\$: variable alphanumérique.

Cette instruction a une double syntaxe. Elle permet dans le premier cas de définir la couleur "c" et le motif du remplissage avec les paramètres "a" et "b".

Le paramètre "a" indique le mode de remplissage :

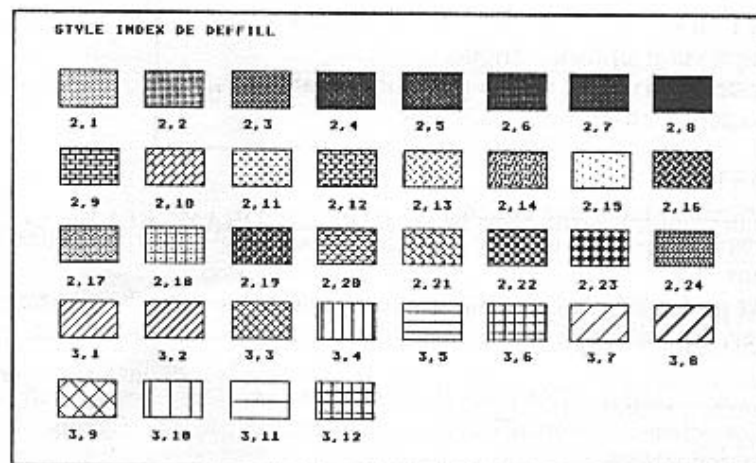
0 = vide ;

1 = plein ;

3 = hachuré ;

4 = défini par l'utilisateur.

Le paramètre "b" spécifie le motif choisi parmi vingt-quatre types dans les pointillés et parmi douze dans les hachurés.



Pour définir un motif propre, il faut constituer une matrice 16x16. La variable alphanumérique "var\$" doit résulter de seize concaténations successives de seize bits transformés en chaînes de caractères par la fonction MKIS en haute résolution (un plan), de trente-deux concaténations en moyenne résolution (deux plans) et de soixante-quatre concaténations en basse résolution (quatre plans).

Les paramètres étant facultatifs, il est indispensable de mettre les virgules séparatrices en lieu et place de ceux qui sont omis.

Voir également les instructions **FILL**, **MKIS**, **PBOX**, **PCIRCLE**, **PELLIPSE** et **POLYFILL**.

DEFLINE [type] [,largeur] [,tdébut] [,tfin]

type,largeur,tdébut,tfin : expressions numériques entières.

Cette instruction spécifie les lignes utilisées par les instructions **BOX**, **CIRCLE**, **DRAW**, **ELLIPSE**, **LINE**, **RBOX** et **POLYLINE**. Le paramètre "type" dé-

termine le type de trait, "largeur" indique la largeur en nombre de points graphiques (pixels), "tdébut" et "tfin" donnent respectivement la forme d'extrémité à utiliser en début et fin de ligne :

0 = normal ;

1 = flèche ;

2 = arrondi.

—————	DEFFILL 1, 1
-----	DEFFILL 2, 1
.....	DEFFILL 3, 1
- - - - -	DEFFILL 4, 1
.....	DEFFILL 5, 1
.....	DEFFILL 6, 1

Voir également les instructions **BOX**, **CIRCLE**, **DRAW**, **ELLIPSE**, **LINE**, **RBOX** et **POLYLINE**.

DEFMARK [c][,symbole][,taille]

c,symbole,taille : expressions numériques entières.

Cette instruction spécifie la couleur "c" le paramètre "symbole" pourra prendre les valeurs suivantes :

1 = point ;

2 = plus ;

3 = étoile ;

4 = rectangle ;

5 = croix ;

6 = losange.

Le paramètre "taille" est la dimension en pixels du symbole utilisé lors de l'instruction **POLYMARK**. La taille peut prendre les valeurs 0, 20, 40, 60, 80 et 100.

Voir également l'instruction **POLYMARK**.

DEFTEXT [c][,type][,alpha][,hauteur]

c,type,alpha,hauteur : expressions numériques entières.

Cette instruction permet de définir la couleur "c", le "type" de caractères :

0 = normal ;

1 = gras ;

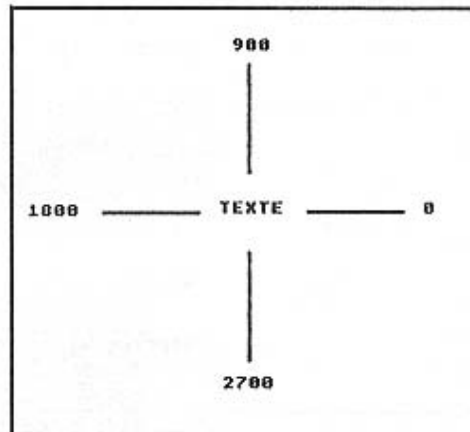
2 = clair ;

4 = italique ;

8 = souligné ;
16 = contour.

Pour avoir plus d'une caractéristique simultanément, il suffit d'additionner les valeurs des types fondamentaux : par exemple, écriture claire et oblique; type sera 4+2=6.

L'angle d'écriture est exprimé en 1/10 de degrés.



Le paramètre "hauteur" détermine la hauteur en pixel des caractères :

4 = caractères contenus dans les icônes ;

6 = caractères fins allongés ;

13 = caractères normaux ;

32 = grands caractères.

Toutes les valeurs sont admises, mais rares sont celles qui permettent lecture aisée.

Voir également les instructions **TEXT** et **COLOR**.

DEFMOUSE var\$

n : expression numérique entière ;

var\$: expression alphanumérique.

Cette instruction définit la forme de la souris :

n = 0 Flèche ;

n = 1 X allongé ;

n = 2 Abeille ;

n = 3 Main avec doigt pointé ;

n = 4 Main plate ;

n = 5 Réticule fin ;

n = 6 Réticule épais ;

n = 7 Réticule avec contour.

Si l'on désire donner même le motif de la souris il suffit de définir var\$ ainsi :

var\$ = MKI\$(coordonnée x du point d'action)

- + MKI\$(coordonnée y du point d'action)
- + MKI\$(1)
- + MKI\$(couleur du masque) généralement 0
- + MKI\$(couleur du curseur) généralement 1
- + M\$ M\$ étant composé de 16 MKI\$
- + C\$ C\$ étant composé de 16 MKI\$

voir MKI\$ pour plus de précision.

DEFNUM n

n : expression numérique entière.

Cette instruction formate les nombres affichés par PRINT. Seuls les n premiers chiffres d'un nombre sont affichés (le point décimal n'est pas pris en compte).

KEYDEF n,ch\$

n : expression numérique entière ;

ch\$: expression alphanumérique.

Cette instruction définit la touche de fonction avec la chaîne de caractère spécifiée d'une longueur maximale de 31 caractères. Les touches F1 à F10 sont respectivement codées de 1 à 10 et de 11 à 20 lorsque la touche <SHIFT> est simultanément pressée. L'affectation de ces touches reste valable sous l'éditeur du GFA Basic toutefois la touche <ALTERNATE> doit également être pressée.

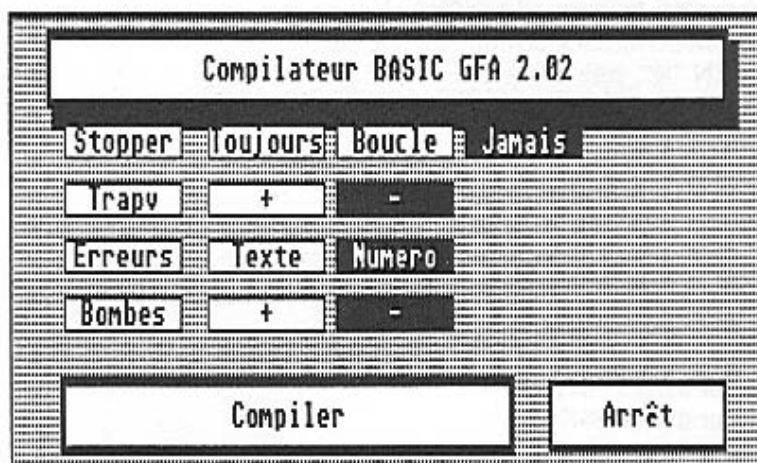
MESSAGES D'ERREURS

Voici la liste complète des message d'erreurs du GFA Basic :

- 0 : division par 0 ;
- 1 : dépassement de capacité ;
- 2 : le nombre n'est pas un Integer -2147483648 .. 2147483647 ;
- 3 : le nombre n'est pas un octet 0 .. 255 ;
- 4 : le nombre n'est pas un mot 0 .. 65535 ;
- 5 : racine carrée d'un nombre négatif impossible ;
- 6 : logarithme d'un nombre inférieur à zéro impossible ;
- 7 : erreur inconnue ;
- 8 : mémoire pleine ;
- 9 : fonction ou instruction impossible ;
- 10 : chaîne trop longue, max 32767 caractères ;
- 11 : le programme n'est pas en GFA BASIC version 2.0 ;
- 12 : programme trop grand, mémoire pleine ;
- 13 : le programme n'est pas en GFA BASIC ;
- 14 : champ dimensionné deux fois ;
- 15 : champ non dimensionné ;
- 16 : index de champ trop grand ;
- 17 : index de dim trop grand ;
- 18 : mauvais nombre d'indices ;
- 19 : procédure introuvable ;
- 20 : label introuvable ;
- 21 : pour OPEN, utiliser 'Input', 'Output', 'Random', 'Append', 'Update' ;
- 22 : fichier déjà ouvert ;
- 23 : mauvais numéro de fichier ;
- 24 : fichier non ouvert ;
- 25 : mauvaise saisie, ce n'est pas un nombre ;
- 26 : fin de fichier atteinte, EOF ;
- 27 : trop de points pour POLYLINE/POLYFILL, max 128 ;
- 28 : le champ ne peut avoir qu'une dimension ;
- 29 : nombre de points plus grand que le champ ;
- 30 : MERGE, ce n'est pas un fichier ASCII ;
- 31 : MERGE, ligne trop longue ;
- 32 : ==> syntaxe incorrecte, arrêt du programme ;
- 33 : marque non définie ;
- 34 : trop peu de données ;
- 35 : donnée non numérique ;
- 36 : erreur de syntaxe dans la donnée, utiliser les "" par paire ;
- 37 : disquette pleine ;
- 38 : instruction impossible en mode direct ;
- 39 : erreur de programmation, GOSUB impossible ;
- 40 : CLEAR n'est pas possible dans une boucle FOR/NEXT ou une procédure ;
- 41 : CONT impossible ;

- 42 : trop peu de paramètres ;
- 43 : expression trop complexe ;
- 44 : fonction indéfinie ;
- 45 : trop de paramètres ;
- 46 : paramètre inexact, ce doit être un nombre ;
- 47 : paramètre inexact, ce doit être une chaîne ;
- 48 : OPEN "R", enregistrement trop long ;
- 49 : trop de fichiers "R" (max 10) ;
- 50 : pas de fichier "R" ;
- 51 : seul un champ est possible avec un OPEN "R" ;
- 52 : champ plus grand que l'enregistrement ;
- 53 : trop de champs (max 19) ;
- 54 : mauvaise longueur d'enregistrement GET/PUT ;
- 55 : mauvais numéro d'enregistrement GET/PUT ;
- 60 : longueur de chaîne de SPRITE erronée ;
- 61 : erreur dans l'exécution de l'instruction RESERVE ;
- 62 : erreur dans MENU ;
- 63 : erreur dans RESERVE ;
- 64 : erreur dans pointeur ;
- 65 : champ supérieur à 256 ;
- 66 : VAR-champ ? ;
- 67 : erreur dans l'exécution des instructions ASIN ou ACOS ;
- 68 : erreur dans la déclaration d'une variable en VAR ;
- 69 : ENDFUNC sans RETURN ;
- 71 : index trop grand ;
- 90 : erreur dans LOCAL ;
- 91 : erreur dans FOR ;
- 92 : RESUME (NEXT) impossible, FATAL, FOR ou LOCAL ;
- 93 : erreur dans la pile ;
- 100 : (c) copyright 1986, GFA Systemtechnik.

COMPILATEUR DU GFA BASIC 2.02



Cette annexe fait référence au compilateur GFA version 2.02. Il est évident que de nouvelles moutures pourraient être disponibles dans les mois qui suivront la parution de cet ouvrage nous ferons donc des additifs dans les éditions futures si cela s'avère nécessaire.

Le compilateur nécessite un programme Basic GFA (.BAS) donc sauvegardé grâce à l'option SAVE. En effet, il utilise le codage interne propre à l'interpréteur. Toutes les instructions présentes sous l'interpréteur du 2.02 sont acceptées sauf celles qui sont spécifiques du mode interprété. En voici la liste :

CONT	PSAVE
DEFLIST	SAVE
LIST	STOP
LLIST	TROFF
LOAD	TRON

La compilation entraîne parfois des nuances dans la signification des instructions, nous les verrons cas par cas.

LES OPTIONS DU COMPILATEUR

L'interpréteur GFA possède une instruction qui permet de configurer directement le compilateur sans avoir à spécifier ces directives lors de la compilation.

Cette instruction se nomme **OPTION**.

OPTION "U?"

Cette option décide de la présence et de l'endroit du test des touches <CONTROL> + <SHIFT> + <ALTERNATE>.

OPTION "U0"

Interdit le test des touches <CONTROL>+<SHIFT>+<ALTERNATE>. Cette option correspond à "STOPPER JAMAIS" dans la fenêtre du compilateur ;

OPTION "U1"

Provoque le test exclusivement à l'endroit du programme où se trouve cette option ;

OPTION "U2"

Effectue un test avant chacune des instructions de contrôle GOTO, LOOP, UNTIL, WEND et NEXT. Cette option correspond à "STOPPER BOUCLE" dans la fenêtre du compilateur ;

OPTION "U3"

Effectue un test après chaque instruction. Cette option correspond à "STOPPER TOUJOURS" dans la fenêtre du compilateur.

La dernière instruction **OPTION "U?"** est celle qui est considérée comme valide jusqu'à ce qu'une nouvelle directive soit sélectionnée.

OPTION "E?"

Cette option décide de l'intégration des messages d'erreur en clair dans le programme compilé.

OPTION "E+"

Intègre les messages d'erreur dans le programme compilé. Cette option correspond à "Erreurs Texte" dans la fenêtre du compilateur ;

OPTION "E-"

N'intègre pas les messages d'erreur dans le programme compilé seuls les numéros des erreurs sont affichés. Cette option correspond à "Erreurs Numéro" dans la fenêtre du compilateur.

La dernière instruction **OPTION "E?"** est celle qui est considérée comme valide jusqu'à ce qu'une nouvelle directive soit sélectionnée.

OPTION "B?"

Cette option décide de l'intégration des routines d'interception des erreurs-bombes dans le programme compilé afin d'avoir une chance de préserver le programme lors d'une erreur-bombe (si celle-ci n'est pas fatale).

OPTION "B+"

Intègre ces routines d'interception dans le programme compilé. Cette option correspond à "Bombes +" dans la fenêtre du compilateur ;

OPTION "B-"

N'intègre pas ces routines d'interception dans le programme compilé. Cette option correspond à "Bombes -" dans la fenêtre du compilateur.

OPTION "T?"

Cette option décide de la présence du test de débordement des instructions arithmétiques entières.

OPTION "T+"

Inclut une instruction TRAPV après chaque instruction concernée. Cette option correspond à "Trapv +" dans la fenêtre du compilateur ;

OPTION "T-"

N'inclut pas cette instruction. Cette option correspond à "Trapv -" dans la fenêtre du compilateur.

INSTRUCTIONS DU GFA 2.0

287

* 46
 ABS 60
 ADD 53, 54
 ADDRIN 252
 ADDROUT 252
 ALERT 187
 AND 37, 57
 ARRAYFILL 43
 ARRPTR 46
 ASC 67
 ATN 60
 BASEPAGE 99
 BGET 207
 BIN\$ 54
 BIOS 219
 BITBLT 200
 BLOAD 207
 BMOVE 99
 BOX 136
 BPUT 207
 BSAVE 207
 C: 117
 CALL 117
 CHAIN 22
 CHDIR 213
 CHDRIVE 213
 CHR\$ 54
 CIRCLE 136
 CLEAR 30
 CLEARW 188
 CLOSE 207
 CLOSEW 209
 CLR ~~45~~ 42
 CLS 25, 188
 COLOR 137
 CONT 30
 CONTRL 253
 COS 60
 CRSCOL 25, 188
 CRSLIN 25, 188
 CVD 41
 CVF 41
 CVI 41
 CVL 41
 CVS 42
 DATA 30
 DATE\$ 105

DEC 54
 DEFFILL 137
 DEFFN/FN 86
 DEFINE 138
 DEFLIST 21
 DEFMARK 138
 DEFMOUSE 188
 DEFNUM 30
 DEFTXT 139
 DFRE 213
 DIM 44
 DIM? 44
 DIR 214
 DIR\$ 214
 DIV 54
 DO/LOOP 86
 DPEEK 48
 DPOKE 48
 DRAW 140
 EDIT 23
 ELLIPSE 141
 END 30
 EOF 209
 EQV 38, 57
 ERASE 44
 ERR 95
 ERROR 95
 EVEN 55
 EXEC 117
 EXIST 209
 EXIT IF 87
 EXP 71
 FALSE 55
 FATAL 95
 FIELD 209
 FILES 214
 FILESELECT 215
 FILL 141
 FIX 61
 FOR/NEXT 87
 FORM INPUT 27
 FRAC 61
 FRE 99
 FULLW 189
 GB 257
 GCONTRL 257
 GEMDOS 236

GEMSYS	257	MID\$	69
GET	141, 209	MIN	62, 69
GIN TIN	257	MKDS	42
GIN TOUT	257	MKDIR	216
GOSUB	88	MKF\$	42
GOTO	88	MKI\$	42
GRAPHMODE	141	MKL\$	42
HARDCOPY	30	MKS\$	42
HEX\$	42	MOD	55
HIDEM	189	MONITOR	117
HIMEM	99	MOUSE	192
IF/ELSE/ENDIF	88	MOUSEK	192
IMP	38, 57	MOUSEX	192
INC	55	MOUSEY	192
INFOW	189	MUL	55, 56
INKEY\$	27	NAME	210
INP	29, 209	NEW	25
INP?	28	NOT	38, 57
INPUT	28, 113	OCT\$	42
INPUT\$	29, 213	ODD	56
INSTR	67	ON BREAK	95
INT	48, 61	ON BREAK CONT	95
INTIN	259	ON BREAK GOSUB	96
INTOUT	259	ON ERROR	96
KILL	209	ON ERROR GOSUB	96
LEFT\$	68	ON GOSUB	89
LEN	68	ON MENU	192
LET	30	ON MENU GOSUB	192
LINE	141	ON MENU BUTTON	192
LINE INPUT	27, 213	ON MENU IBOX	193
LIST	23	ON MENU KEY	193
LLIST	23	ON MENU MESSAGE	193
LOAD	23	ON MENU OBOX	193
LOC	209	OPEN	210
LOCAL	89	OPENW	211
LOF	209	OPTION BASE	30
LOG	62	OR	37, 57
LOG10	62	OUT	28, 210
LPEEK	48	OUT?	28
LPOKE	48	PAUSE	30
LPOS	27	PBOX	142
LPRINT	27	PCIRCLE	142
LSET	68, 209	PEEK	48
MAX	62, 68	PELLIPSE	142
MENU	189, 200	PI	62
MENU KILL	200	PLOT	142
MENU OFF	200	POINT	142

POKE 48	STR\$ 64
POLYFILL 142	STRING\$ 71
POLYLINE 143	SUB 56
POLYMARK 143	SWAP 43, 45, 59
POS 28	SYSTEM 24
PRBOX 143	TAB 29
PRINT 28, 212	TAN 64
PRINT USING 28, 212	TEXT 145
PROCEDURE/RETURN 89	TIMES 106
PSAVE 23	TIMER 106
PTSIN 264	TITLEW 194
PTSOUT 264	TROFF 96
PUT 143, 211	TRON 96
QUIT 24	TRUE 56
RANDOM 63	TRUNC 64
RBOX 144	TYPE 49
READ 33	UPPER\$ 72
RELSEEK 211	VAL 42
REM 33	VAL? 43
REPEAT/UNTIL 89	VARPTR 50
RESERVE 101	VDIBASE 266
RESTORE 33	VDISYS 266
RESUME 96	VOID 33
RETURN 90	VSYNC 145
RIGHT\$ 70	VTAB 29
RMDIR 216	WAVE 113
RND 63	WHILE/WEND 91
RSET 70, 211	WINTAB 194
RUN 24	WRITE 29
SAVE 24	XBIOS 224
SDPOKE 49	XOR 36, 58
SEEK 211	
SETCOLOR 144	
SETTIME 105	
SGET 145	
SGN 62	
SHOWM 193	
SIN 64	
SLPOKE 49	
SOUND 113	
SPACE\$ 71	
SPC 71	
SPOKE 49	
SPRITE 145	
SPUT 145	
SQR 64	
STOP 33	

INDEX ALPHABETIQUE

293

~ 31
 * 45
 ABS 60
 ABSOLUTE 46
 ACHAR 181
 ACLIP 181
 ACOS 59
 ADD 53, 54
 ADDRIN 252
 ADDRROUT 252
 AFTER 105
 AFTER CONT 105
 AFTER STOP 105
 ALERT 187
 ALINE 181
 APOLY 181
 APPL_EXIT 253
 APPL_FIND 253
 APPL_INIT 253
 APPL_READ 253
 APPL_TPLAY 253
 APPL_TRECORD 253
 APPL_WRITE 253
 ARECT 182
 ARRAYFILL 43
 ARRPTR 46
 ASC 67
 ASIN 60
 ATEXT 182
 ATN 60
 BASEPAGE 99
 BCHG 59
 BCLR 59
 BGET 207
 BIN\$ 54
 BIOS 219
 BITBLT 182
 BLOAD 207
 BMOVE 99
 BOUNDARY 136
 BOX 136
 BPUT 207
 BSAVE 207
 BSET 59
 BTST 59
 BYTE 44, 45, 59
 C: 117
 CALL 117
 CARD 45, 47
 CFLOAT 39
 CHAIN 22
 CHAR 45
 CHDIR 213
 CHDRIVE 213
 CHR\$ 54
 CINT 41
 CIRCLE 136
 CLEAR 30
 CLEARW 188
 CLOSE 207
 CLOSEW 209
 CLR 42
 CLS 25, 188
 COLOR 137
 CONT 30
 CONTRL 253
 COS 60
 COSQ 61
 CRSCOL 25, 188
 CRSLIN 25, 188
 CVD 39
 CVF 39
 CVI 39
 CVL 39
 CVS 40
 DATA 30
 DATE\$ 105
 DEC 54
 DEFBIT 21
 DEFBYT 21
 DEFFILL 137
 DEFFLT 21
 DEFFN/FN 86
 DEFINT 21
 DEFLINE 138
 DEFLIST 21
 DEFMARK 138
 DEFMOUSE 188
 DEFNUM 30
 DEFSTR 22
 DEFTXT 139
 DEFWRD 22

DEG 61
 DELAY 30
 DELETE 41, 42
 DFRE 213
 DIM 42
 DIM? 42
 DIR 214
 DIR\$ 214
 DIV 54
 DO/LOOP 86
 DO/UNTIL 86
 DO/WHILE 86
 DOUBLE 46
 DPEEK 47
 DPOKE 47
 DRAW 140
 DUMP 95
 EDIT 23
 ELLIPSE 141
 ELSEIF 86
 END 30
 EOF 208
 EQV 38, 57
 ERASE 42
 ERR 95
 ERR\$ 95
 ERROR 95
 EVEN 55
 EVERY 105
 EVERY CONT 105
 EVERY STOP 105
 EVNT_BUTTON 253
 EVNT_DCLIC 254
 EVNT_KEYBD 254
 EVNT_MESAG 254
 EVNT_MOUSE 254
 EVNT_MULTI 255
 EVNT_TIMER 255
 EXEC 117
 EXIST 208
 EXIT IF 87
 EXP 61
 FALSE 55
 FATAL 95
 FGETDTA 214
 FIELD 209
 FILES 214
 FILESELECT 215
 FILL 141
 FIX 61
 FLOAT 47
 FOR/NEXT 87
 FORM INPUT 25
 FORM_ALERT 255
 FORM_BUTTON 255
 FORM_CENTER 255
 FORM_DIAL 256
 FORM_DO 256
 FORM_ERROR 256
 FORM_KEYBD 256
 FRAC 61
 FRE 99
 FSEL_INPUT 256
 FSETDTA 215
 FSFIRST 215
 FSNEXT 216
 FULLW 189
 FUNCTION/ENDFUNC 88
 GB 256
 GCONTRL 257
 GDOS? 257
 GEMDOS 236
 GEMSYS 257
 GET 141, 208
 GINTIN 257
 GINTOUT 257
 GOSUB 88
 GOTO 88
 GRAF_DRAGBOX 257
 GRAF_GROWBOX 257
 GRAF_HANDLE 258
 GRAF_MKSTATE 258
 GRAF_MOUSE 258
 GRAF_MOVEBOX 258
 GRAF_RUBBERBOX 258
 GRAF_SHRINKBOX 258
 GRAF_SLIDEBOX 258
 GRAF_WATCHBOX 259
 GRAPHMODE 141
 HARDCOPY 30
 HEX\$ 42
 HIDEM 189
 HIMEM 99
 HLINE 184

HTAB 25
 IF/ELSE/ENDIF 88
 IMP 38, 57
 INC 55
 INFOW 189
 INKEY\$ 27
 INLINE 100
 INP 26, 209
 INP? 26
 INPAUX 26
 INPUT 26, 213
 INPUT\$ 27, 213
 INSERT 42
 INSTR 67
 INT 48, 61
 INTIN 259
 INTOUT 259
 KEYDEF 131
 KEYGET 131
 KEYLOOK 131
 KEYPAD 131
 KEYPRESS 131
 KEYTEST 131
 KILL 209
 L=A 183
 LEFT\$ 68
 LEN 68
 LET 30
 LINE 141
 LINE INPUT 27, 213
 LIST 23
 LLIST 23
 LOAD 23
 LOC 209
 LOCAL 89
 LOCATE 27
 LOF 209
 LOG 62
 LOG10 62
 LONG 48
 LPEEK 48
 LPOKE 48
 LPOS 27
 LPRINT 27
 LSET 68, 209
 MALLOC 100

MAX 62, 68
 MENU 189, 200
 MENU KILL 200
 MENU OFF 200
 MENU_BAR 259
 MENU_ICHECK 259
 MENU_IENABLE 259
 MENU_REGISTER 259
 MENU_TEXT 259
 MENU_TNORMAL 259
 MFREE 100
 MID\$ 69
 MIN 62, 69
 MKD\$ 42
 MKDIR 216
 MKF\$ 42
 MKI\$ 42
 MKL\$ 42
 MKS\$ 42
 MOD 55
 MONITOR 117
 MOUSE 192
 MOUSEK 192
 MOUSEX 192
 MOUSEY 192
 MSHRINK 100
 MUL 55, 56
 NAME 210
 NEW 23
 NOT 36, 57
 OB_ADR 260
 OB_FLAGS 260
 OB_H 260
 OB_HEAD 260
 OB_NEXT 260
 OB_SPEC 260
 OB_STATE 261
 OB_TAIL 261
 OB_TYPE 261
 OB_W 262
 OB_X 262
 OB_Y 262
 OBJC_ADD 262
 OBJC_CHANGE 262
 OBJC_DELETE 262
 OBJC_DRAW 263

OBJC_EDIT 263
 OBJC_FIND 263
 OBJC_OFFSET 263
 OBJC_ORDER 263
 OCT\$ 42
 ODD 56
 ON BREAK 95
 ON BREAK CONT 95
 ON BREAK GOSUB 96
 ON ERROR 96
 ON ERROR GOSUB 96
 ON GOSUB 89
 ON MENU 192
 ON MENU GOSUB 192
 ON MENU BUTTON 192
 ON MENU IBOX 193
 ON MENU KEY 193
 ON MENU MESSAGE 193
 ON MENU OBOX 193
 OPEN 210
 OPENW 211
 OPTION BASE 30
 OR 35, 57
 OUT 28, 210
 OUT? 28
 PAUSE 30
 PBOX 142
 PCIRCLE 142
 PEEK 48
 PELLIPSE 142
 PI 62
 PLOT 142
 POINT 142
 POKE 48
 POLYFILL 142
 POLYLINE 143
 POLYMARK 143
 POS 28
 PRBOX 143
 PRED 62, 69
 PRINT 28, 212
 PRINT USING 28, 212
 PROCEDURE/RETURN 89
 PSAVE 23
 PSET 184
 PTSIN 264
 PTSOUT 264
 PTST 184
 PUT 143, 211
 QSORT 44
 QUIT 24
 RAD 62
 RAND 63
 RANDOM 63
 RANDOMIZE 63
 RBOX 144
 RC_COPY 264
 RC_INTERSECT 264
 RCALL 117
 READ 31
 RECALL 211
 RECORD 211
 RELSEEK 211
 REM 31
 REPEAT/UNTIL 89
 RESERVE 101
 RESTORE 31
 RESUME 96
 RETURN 90
 RIGHT\$ 70
 RINSTR 70
 RMDIR 216
 RND 63
 ROL 58
 ROR 58
 ROUND 63
 RSET 70, 211
 RSRC_FREE 265
 RSRC_GADDR 264
 RSRC_LOAD 265
 RSRC_OBFIX 265
 RSRC_SADDR 265
 RUN 24
 SAVE 24
 SCRP_READ 265
 SCRP_WRITE 265
 SDPOKE 49
 SEEK 211
 SELECT/ENDSELECT 90
 SETCOLOR 144
 SETDRAW 145
 SETMOUSE 193
 SETTIME 105
 SGET 145

SGN 63
 SHEL_ENVRN 265
 SHEL_FIND 265
 SHEL_GET 265
 SHEL_PUT 266
 SHEL_READ 266
 SHEL_WRITE 266
 SHL 58
 SHOWM 193
 SHR 58
 SIN 64
 SINGLE 49
 SINQ 64
 SLPOKE 49
 SOUND 113
 SPACE\$ 71
 SPC 71
 SPOKE 49
 SPRITE 145
 SPUT 145
 SQR 64
 SSORT 45
 STICK 109
 STOP 33
 STORE 212
 STR\$ 64
 STRIG 109
 STRING\$ 71
 SUB 56
 SUCC 64, 71
 SWAP 43, 45, 59
 SYSTEM 24
 TAB 29
 TAN 64
 TEXT 145
 TIMES\$ 106
 TIMER 106
 TITLEW 194
 TOPW 194
 TOUCH 212
 TRACE\$ 96
 TRIM\$ 72
 TROFF 96
 TRON 96
 TRUE 56
 TRUNC 64
 TYPE 49

UPPER\$ 72
 V: 50
 V_CLRWK 266
 V_CLRWK 266
 V_CLSVWK 266
 V_CLSWK 266
 V_OPNVWK 267
 V_OPNWK 267
 V_UPDWK 267
 V=H 266
 VAL 42
 VAL? 43
 VAR 90
 VARPTR 50
 VDIBASE 266
 VDISYS 266
 VOID 31
 VQT_EXTEND 267
 VQT_NAME 267
 VSETCOLOR 146
 VST_LOAD_FONTS 267
 VST_UNLOAD_FONTS 267
 VSYNC 145
 VTAB 29
 W_HAND 195
 W_INDEX 195
 WAVE 113
 WHILE/WEND 91
 WIND_CALC 268
 WIND_CLOSE 268
 WIND_CREATE 268
 WIND_DELETE 268
 WIND_FIND 268
 WIND_GET 269
 WIND_OPEN 269
 WIND_SET 269
 WIND_UPDATE 260
 WINTAB 194
 WORD 59
 WORK_OUT 270
 WRITE 29
 XBIOS 224
 XOR 36, 58

CONSEILS DE LECTURE

L'assembleur 68000 de l'Atari ST par Olivier Hard (Editions Cedric Nathan)

Cet ouvrage vous présente le microprocesseur 68000 en détail : jeu d'instructions, adressage, implantation en mémoire sur les modèles Atari. Les routines du système d'exploitation GEM sont ensuite décrites et utilisées dans plusieurs programmes d'application. Cet ouvrage, construit d'une façon très progressive, intéressera tous les programmeurs sur Atari, débutants ou chevronnés.

Super Jeux Atari ST, Basic GFA par Jean-François Sehan (Editions PSI)

Cet ouvrage qui présente 50 programmes de jeux d'adresse, de réflexion et de hasard, permet au lecteur de maîtriser rapidement le Basic GFA de l'Atari ST. Celui-ci apprend, en jouant, à construire des programmes de plus en plus complexes en s'aidant des commentaires pédagogiques de l'auteur et de sa précise liste de variables. Ce livre contient des programmes faciles à rentrer et à modifier et est parfaitement adapté aux possibilités de l'Atari.

L'Atari ST en action par David Lawrence et Mark England (Editions Edimicro)

Ce livre présente toutes les caractéristiques du ST, et explique comment exploiter au mieux ses ressources. Chacun des aspects est traité complètement, de la façon la plus claire : l'unité centrale, le sous-système graphique, le sous-système de musique, les périphériques, etc. Les points importants font l'objet d'un chapitre complet : le gestionnaire de graphiques GEM, le système d'exploitation TOS, le plan de travail et sa configuration, les dossiers. Il apprend au lecteur à se servir efficacement des progiciels d'application, dans tous les domaines.

Peintre et musicien sur Atari ST par Daniel-Jean David (Editions Edimicro)

Cet ouvrage s'adresse à tous les passionnés, qui veulent utiliser pleinement les ressources graphiques et sonores de leur Atari ST. Il présente, de façon claire et précise, à l'aide d'exemples et de nombreuses illustrations et figure : les modes graphiques, les graphiques de gestion (courbes, diagrammes en bâtons, etc), les graphiques artistiques et scientifiques (dessins en trois dimensions, perspectives et rotations), les lutins (objets graphiques déplaçables) et les sons (bruits, imitation d'instruments, partitions, etc).

Clefs pour Atari ST - Nouvelle édition
par Franck-Olivier Lelaidier (Editions PSI)

Clefs pour Atari ST s'adresse à deux types de personnes : les programmeurs sur Atari ST 520, 1040 ou Méga ST, mais aussi les utilisateurs avertis désirant comprendre et mieux utiliser leur machine. En une dizaine de chapitres, l'auteur fait un tour d'horizon complet de tous les sujets intéressant le programmeur. Enfin réunies dans un seul et même ouvrage, présentées de façon claire et synthétique, vous trouverez dans ce livre toutes les informations dont vous avez besoin.

Atari ST efficace
par Christophe Castro et Augustin Garcia Ampudia (Editions PSI)

Comment choisir une forme de souris sur votre Atari ST ? Peut-on programmer le clavier ? Qu'est-ce que le bureau de GEM ? Quel écran pour votre ordinateur ? Comment changer la résolution de votre écran ? Comment lire les disquettes d'autres ordinateurs ? Qu'est-ce qu'un objet ? En quoi consiste la copie avec un Ram-disque ? Comment installer une imprimante ? Peut-t-on sélectionner des objets dans une fenêtre inactive ? Savez-vous installer un accessoire sur votre Atari ST ? Cet ouvrage apporte des réponses claires et précises à toutes ces questions et permet de découvrir les mille et une astuces d'utilisation de l'Atari ST.

● Achevé d'imprimer
par l'imprimerie Tardy Quercy S.A. - Cahors
Dépôt légal : avril 1989 - N° d'imprimeur : 90320 F
N° d'édition : 86595-557-1
ISBN 2-86595-557-5

P.S.I

CLEFS
POUR

GFA BASIC 2 et 3 SUR ATARI

Le Basic GFA est devenu un des standards de programmation sur la gamme Atari. Apprécié de tous les programmeurs, le Basic GFA met à leur disposition des instructions et fonctions évoluées dans tous les domaines tout en offrant une grande rapidité et une simplicité d'utilisation.

Clefs pour GFA Basic 2 et 3 sur Atari est un ouvrage de référence complet qui vous apportera une information claire et précise sur tous les aspects du Basic GFA.

Vous découvrirez successivement l'éditeur, les commandes et instructions de base, les différents types d'opérateurs, les variables, tableaux et pointeurs, les fonctions numériques, le traitement des chaînes de caractères, les instructions de structure, la gestion des erreurs, la gestion de la mémoire, l'horloge et les interruptions, la gestion du joystick et des sons, la communication avec d'autres langages, la gestion du clavier, le graphisme, les fenêtres, les menus, la souris, les fichiers et répertoires, le BIOS, XBIOS, et GEMDOS, etc.

De nombreuses annexes terminent ce livre et en font un guide complet pour le programmeur en Basic GFA sur Atari.



9 782865 955572

PRIX : 135 FF
ISBN : 2-86595-557-5
500523

DIFFUSE PAR P.C.V. DIFFUSION
9, RUE MECHAIN
75680- PARIS CEDEX 14